

NoSQL

Database for Storage and
Retrieval of Data in Cloud

Edited by
Ganesh Chandra Deka

NoSQL

**Database for Storage and
Retrieval of Data in Cloud**



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

NoSQL

Database for Storage and Retrieval of Data in Cloud

Edited by
Ganesh Chandra Deka



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2017 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper

International Standard Book Number-13: 978-1-4987-8436-8 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Names: Deka, Ganesh Chandra, 1969- author.
Title: NoSQL : database for storage and retrieval of data in cloud / Ganesh Chandra Deka.
Description: Boca Raton, FL : CRC Press, Taylor & Francis Group, [2016] | Includes bibliographical references and index.
Identifiers: LCCN 2016039389| ISBN 9781498784368 (hardback : alk. paper) | ISBN 9781498784375 (ebook)
Subjects: LCSH: Cloud computing. | Non-relational databases. | Database design. | SQL (Computer program language)
Classification: LCC QA76.585 .D45 2016 | DDC 004.67/82--dc23
LC record available at <https://lccn.loc.gov/2016039389>

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

Preface.....	vii
Editor.....	ix
Contributors.....	xi
1. Distributed Transaction Processing	1
<i>Rebika Rai</i>	
2. X/Open Distributed Transaction Processing Model Using EJB and MTS	23
<i>S. Gunasekaran and V. Bargavi</i>	
3. Data Migration Techniques from SQL to NoSQL	47
<i>Mydhili K. Nair, Nihal Nayak, and Arjun Rao</i>	
4. Comparative Study on Mostly Used NoSQL Databases	75
<i>K. G. Srinivasa, Nabeel Siddiqui, and Abhishek Kumar</i>	
5. NoSQL and Cloud Paradigm: Characteristics and Classification, Data Storage Technology, and Algorithms	99
<i>Ajanta Das, Anindita Desarkar, and Mridul Paul</i>	
6. A Scalable Record Linkage Technique for NoSQL Databases Using GPGPU	119
<i>Parijat Shukla and Arun K. Somani</i>	
7. NoSQL for Handling Big and Complex Biological Data	143
<i>Awanish Kumar</i>	
8. Applications of Hadoop Ecosystems Tools	159
<i>Pratik Mishra, Mayank Mishra, and Arun K. Somani</i>	
9. Hadoop Ecosystem Tools and Algorithms	177
<i>Kevin Berwind, Marco Xaver Bornschlegl, Michael Kaufmann, Felix Engel, Michael Fuchs, Dominic Heutelbeck, and Matthias Hemmje</i>	
10. Big Data Management Tools for Hadoop	199
<i>V. P. Lijo, Lydia J. Gnanasigamani, Hari Seetha, and B. K. Tripathy</i>	
11. Realization of Optimized Clustering Algorithm on RHadoop	215
<i>Poonam Ghuli, Raksha Shenoy, Shankararama Sharma R, and Rajashree Shettar</i>	
12. Security and Privacy: Challenges and Defending Solutions for NoSQL Data Stores	237
<i>Sandhya Aneja and Nagender Aneja</i>	

13. Challenges and Security Issues of Distributed Databases.....	251
<i>Neha Gupta and Rashmi Agrawal</i>	
14. Security Issues and Privacy Challenges of NoSQL Databases.....	271
<i>Mohammad Samadi Gharajeh</i>	
15. Attack Graph Generation and Analysis Using Graph Database	291
<i>Mridul Sankar Barik, Anirban Sengupta, and Chandan Mazumdar</i>	
16. Hands-On Aerospike.....	311
<i>Khaleel Ahmad and Afsar Kamal</i>	
17. Hands-On Cassandra for Windows.....	323
<i>Khaleel Ahmad and Abdul Haseeb</i>	
18. Hands-On Cloudant.....	333
<i>Masroor Ansari and Khaleel Ahmad</i>	
19. Hands-On InfluxDB.....	341
<i>Khaleel Ahmad and Masroor Ansari</i>	
20. Hands-On Redis	355
<i>Khaleel Ahmad and Mohd Javed</i>	
21. Hands-On RethinkDB.....	365
<i>Dildar Hussain and Khaleel Ahmad</i>	
22. Hands on Neo4j: Graphs for Real-World Applications.....	377
<i>Siddhartha Duggirala</i>	
23. Tutorial on MongoDB.....	401
<i>Prashanta Kumar Das</i>	
24. Introduction to Oracle NoSQL Database.....	405
<i>Suresh Mohan and Hari Seetha</i>	
25. Hosting and Delivering Cassandra NoSQL Database via Cloud Environments.....	415
<i>Skylab Reddy and Pethuru Raj</i>	
Index.....	435

Preface

Large-scale and high-concurrency applications and search engines have appeared to be facing challenges in using the relational database to store and query dynamic user data known as Big Data. NoSQL has emerged as an advanced database technology capable of meeting these requirements. Some experts classify them according to their data model such as aggregate oriented and nonaggregate oriented, while others outline them as four major types, that is, key-value, document databases, column-oriented databases, and graph databases.

Big Data needs innovative technology to extract the hidden potentials on them which are unlimited. One of such Big Data technologies is Apache Hadoop, an open-source software framework. Apache Hadoop and its modules and components such as HDFS, Storm, Zeppelin, HBase, and Hive are capable to handle large amounts of data. New methodologies like IVIS4BigData or CRISP4BigData are available to handle the information visualization from raw data into visual views and to deal with analysis processes and projects, beginning with the business process understanding to *Archiving* and *Retention*. Another such Big Data crunching application is next-generation sequencing (NGS) which is capable of dealing with terabytes or petabytes of genome data requiring high computational power. Furthermore, success of Cassandra for Facebook, Google BigTable for Google, Dynamo for Amazon, MongoDB for eBay, and many other NoSQL tools has shown great performance for practical scenarios where data are continuously growing.

Lots of applications are already migrated from relational database systems to NoSQL database since NoSQL is capable of supporting unstructured text and multiple data structures, rapid change over time, and flexible schema pattern. As a semistructured source, NoSQL can represent data as XML data, and it can be accessed in a hybrid manner like SQL/XML language. As a result, data migration from SQL to NoSQL and vice versa is a very interesting area of research in database technology. There are various techniques of migrating data from the traditional relational SQL-based databases to NoSQL databases. There are translators such as Mongify to migrate data from SQL to MongoDB. Similarly, the tools such as *Batch Importer*, *GEOFF*, or *REST batch API* can be used for migrating (translation) data from MySQL to Neo4j Server, which is a good example of SQL–NoSQL interoperability.

As increasingly sensitive data are being stored in NoSQL databases, security of database from intruders becomes a growing concern. Security problems of NoSQL database environments are categorized into multiple fields including safeguarding integrity, shared data, and compromised clients. Maturing technology of NoSQL databases poses a tremendous opportunity for its adoption in the domain of network security. Research efforts to address the issues related to data protection, user authentication, etc. and to improve the built-in security mechanisms of the NoSQL databases are the need of the hour.

This book will deliberate upon the various aspects of NoSQL databases starting with Distributed Transaction Processing to various tools and technology used for storage of unstructured and semistructured data, contributed by renowned scholars from various countries.

Ganesh Chandra Deka



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Editor

Ganesh Chandra Deka is the Deputy Director (Training) at the Directorate General of Training in the Indian government's Ministry of Skill Development and Entrepreneurship. He is the author of two books in Cloud Computing and coauthor and editor of several books on Big Data Analytics and computer science.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Contributors

Rashmi Agrawal

Faculty of Computer Applications
Manav Rachna International University
Faridabad, India

Khaleel Ahmad

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

Nagender Aneja

Innovation and Enterprise Office
Universiti Brunei Darussalam
Brunei Darussalam

Sandhya Aneja

Faculty of Integrated Technologies
Universiti Brunei Darussalam
Brunei Darussalam

Masroor Ansari

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

V. Bargavi

Department of Computer Science and
Engineering
Coimbatore Institute of Engineering and
Technology
Coimbatore, India

Mridul Sankar Barik

Department of Computer Science and
Engineering
Jadavpur University
Kolkata, India

Kevin Berwind

Faculty of Mathematics and Computer
Science
University of Hagen
Hagen, Germany

Marco Xaver Bornschlegl

Faculty of Mathematics and Computer
Science
University of Hagen
Hagen, Germany

Ajanta Das

Department of Computer Science and
Engineering
Birla Institute of Technology
Kolkata, India

Prashanta Kumar Das

ITI Dhansiri
Assam, India

Anindita Desarkar

Department of Computer Science and
Engineering
Birla Institute of Technology
Kolkata, India

Siddhartha Duggirala

Bharat Petroleum
Mumbai, India

Felix Engel

Faculty of Mathematics and Computer
Science
University of Hagen
Hagen, Germany

Michael Fuchs

Wilhelm Büchner Distance University of
Applied Science
Pfungstadt, Germany

Mohammad Samadi Gharajeh

Young Researchers and Elite Club
Islamic Azad University
Tabriz, Iran

Poonam Ghuli

Department of Computer Science and
Engineering
R.V. College of Engineering
Bengaluru, India

Lydia J. Gnanasigamani

School of Computing Science and
Engineering
VIT University
Vellore, India

S. Gunasekaran

Department of Computer Science and
Engineering
Coimbatore Institute of Engineering and
Technology
Coimbatore, India

Neha Gupta

Faculty of Computer Applications
Manav Rachna International University
Faridabad, India

Abdul Haseeb

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

Matthias Hemmje

Faculty of Mathematics and Computer
Science
University of Hagen
Hagen, Germany

Dominic Heutelbeck

FTK—Research Institute for
Telecommunication and Cooperation
Dortmund, Germany

Dildar Hussain

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

Mohd Javed

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

Afsar Kamal

School of CS and IT
Maulana Azad National Urdu University
Hyderabad, India

Michael Kaufmann

School of Engineering and Architecture
Lucerne University of Applied Sciences
and Arts
Horw, Switzerland

Abhishek Kumar

Department of Computer Science
M. S. Ramaiah Institute of Technology
Bengaluru, India

Awanish Kumar

Department of Biotechnology
National Institute of Technology (NIT)
Chhattisgarh, India

V. P. Lijo

School of Computing Science and
Engineering
VIT University
Vellore, India

Chandan Mazumdar

Department of Computer Science and
Engineering
Jadavpur University
Kolkata, India

Mayank Mishra

Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa

Pratik Mishra

Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa

Suresh Mohan

Server Technologies Curriculum
Oracle
Bengaluru, India

Mydhili K. Nair

Department of Information Science and
Engineering
M. S. Ramaiah Institute of Technology
Bengaluru, India

Nihal Nayak

Department of Information Science and
Engineering
M. S. Ramaiah Institute of Technology
Bengaluru, India

Mridul Paul

Department of Computer Science and
Engineering
Birla Institute of Technology
Kolkata, India

Rebika Rai

Department of Computer Applications
Sikkim University
Sikkim, India

Pethuru Raj

Global Technology Services (GTS) Labs
IBM India
Bengaluru, India

Arjun Rao

Department of Information Science and
Engineering
M. S. Ramaiah Institute of Technology
Bengaluru, India

Skylab Reddy

Global Technology Services (GTS) Labs
IBM India
Bengaluru, India

Hari Seetha

School of Computing Science and
Engineering
VIT University
Vellore, India

Anirban Sengupta

Department of Computer Science and
Engineering
Jadavpur University
Kolkata, India

Shankararama Sharma R

Department of Computer Science and
Engineering
R.V. College of Engineering
Bengaluru, India

Raksha Shenoy

Department of Computer Science and
Engineering
R.V. College of Engineering
Bengaluru, India

Rajashree Shettar

Department of Computer Science and
Engineering
R.V. College of Engineering
Bengaluru, India

Parijat Shukla

Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa

Nabeel Siddiqui

Department of Computer Science
M. S. Ramaiah Institute of Technology
Bengaluru, India

Arun K. Somani

Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa

K. G. Srinivasa

Department of Computer Science
M. S. Ramaiah Institute of
Technology
Bengaluru, India

B. K. Tripathy

School of Computing Science and
Engineering
VIT University
Vellore, India



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

1

Distributed Transaction Processing

Rebika Rai

CONTENTS

1.1	Introduction to Distributed Database.....	1
1.1.1	Distributed Processing and Distributed Database	2
1.1.2	Parallel DBMS and DDBMS.....	3
1.1.3	Distributed Database Techniques.....	4
1.1.4	Concurrency Control in Distributed Database	6
1.1.5	Promises of DDBMS	9
1.2	Introduction to Distributed Transaction Processing.....	10
1.2.1	Background of Distributed Transaction Processing.....	10
1.2.2	Introduction to Distributed Transaction Processing Models.....	11
1.2.2.1	Atomic Actions and Flat Transactions.....	12
1.2.2.2	Nested Transactions	12
1.2.3	Distributed Transaction Processing in Relational and Non-Relational Database	13
1.2.3.1	Distributed Transaction Processing in Relational Database	14
1.2.3.2	Distributed Transaction Processing in Non-Relational Database	15
1.3	Return of ACID Property in Distributed Transaction Processing.....	16
1.3.1	Introduction to ACID Property.....	16
1.3.2	ACID Property and Non-Relational Database	17
1.4	NoSQL in Distributed Transaction Processing	18
1.5	Security Issues in Distributed Transaction Processing Systems	21
	References.....	22

1.1 Introduction to Distributed Database

With the expansion of huge quantity of real-time data, the dimensions of data are escalating exponentially with each passing day, thereby seeking for a centralized repository that can efficiently store the data, which needs to be retrieved, manipulated, and updated using some form of management system. The transaction management module is a very popular component for the management of large collections that guarantee the consistency of data records when multiple users perform concurrent operations on them. The development of database management system (DBMS) helped to fully achieve data independence (transparency) providing centralized controlled data maintenance and access. However, the necessity to balance the computer's workload to avoid peak-load problems when people across an organization want to use it limits users' flexibility in doing their own work

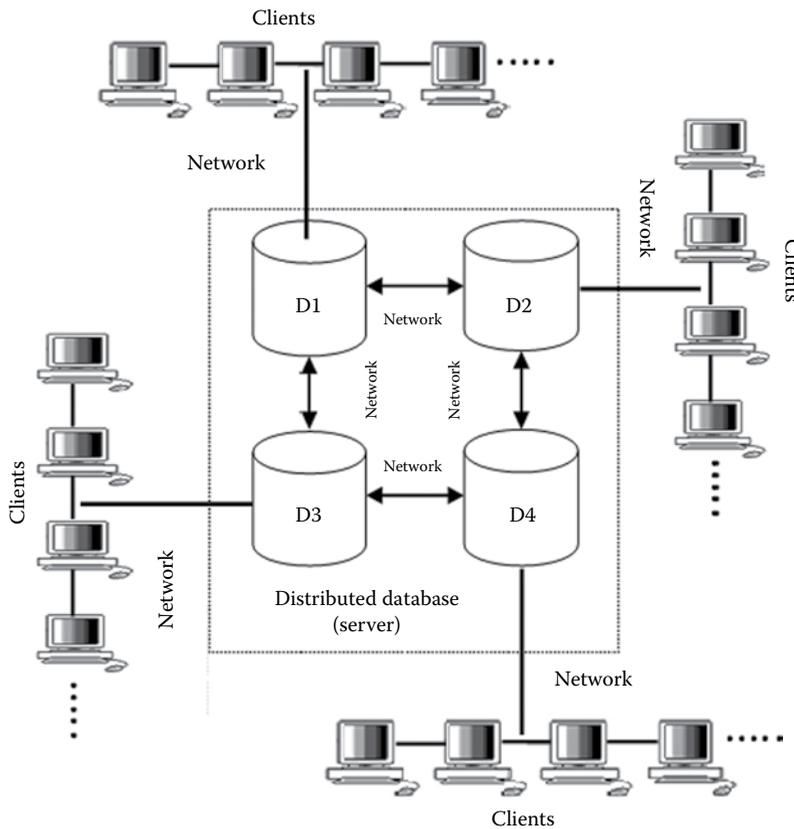


FIGURE 1.1
Components of distributed database.

in a centralized computing system, which has led to the swing toward decentralized/distributed computing.

A distributed database consists of a set of interrelated databases stored on several computers distributed over a network wherein the data can be concurrently accessed and altered. The components of a distributed database include a database server and a client as depicted in [Figure 1.1](#).

A database server is the software that administers a database, and a client is an application that requests information and seeks services from a server. Each computer in a system is represented as a node, and node in a distributed database system can be a client, a server, or both depending on the scenario taken into consideration. Each database server in the distributed database is managed by its local DBMS, and each cooperates to preserve the consistency of the global database. A software system known as distributed database management system (DDBMS) manages a distributed database and makes the distribution apparent to users. It consists of a logical database that is partitioned into a number of sections, and each is stored on one or more computer systems.

1.1.1 Distributed Processing and Distributed Database

Distributed processing refers to the use of more than one computer (or processor) to run an application and perform the processing for an individual task. More often, distributed

processing refers to local area networks (LANs) designed so that a single program can run simultaneously at various sites. Most distributed processing systems contain a sophisticated software that identifies idle CPUs on the network and parcels out programs to make use of them. Distributed processing is composed of distributed databases, wherein the data are stored across computer systems, generally two or more in number. The database system generally keeps track of the location of data so that the distributed character of the database is not evident to users (Manpreet Kaur 2014).

The main goal of a distributed processing system is to connect users and resources in a transparent, open, and scalable way. Ideally, the arrangement in distributed processing system is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems. Resource sharing, scalability, fault tolerance/robustness, and performance/speed are other benefits of distributed processing.

1.1.2 Parallel DBMS and DDBMS

Despite the explosion in terms of speed, computers are not able to keep up with the scale of data becoming available. Manufacturers have come up with the solution known as multiple processors to outwit physical and mechanical limitations on speed of individual processor. The availability of more than two processors definitely boosts up the speed of program getting executed drastically, wherein when one processor is performing a particular aspect of some computation, other computation can be performed by other available processors and the work will advance in parallel. In order to be able to work together, several processors need to be able to share information with each other. Parallel processing can be considered a subset of distributed computing, wherein a single computer uses more than one CPU to execute programs (Manpreet Kaur 2014). A distributed system is a network of independent computers that interact with each other in order to achieve a goal and do not physically share memory or processors. They communicate with each other via *messages*, which are basically a piece/pieces of information transmitted from one computer to another over a network. Messages used for communication can project many things such as the following: one computer can convey to other to execute procedures/functions with particular arguments; to send and accept packets of data; or to propel signals that notify other computers to act in a certain way. The differences between parallel DBMS and DDBMS have been highlighted in [Table 1.1](#) ([Figure 1.2](#)).

Parallel database management refers to the management of data in a tightly coupled multiprocessor computer and is done by a full-fledged DBMS. The basic standard employed by parallel DBMS is to partition the data across multiple nodes, in order to amplify

TABLE 1.1

Parallel DBMS versus DDBMS

Parallel DBMS	DDBMS
Machines are physically located close to each other, for example, same server room.	Machines can be located far-off from each other, for example, in diverse continent.
Machines connect with dedicated high-speed LANs and switches.	Machines can be connected using public-purpose network, for example, Internet.
Communication expenditure is assumed to be small.	Communication expenditure and predicaments cannot be ignored.
Can employ shared-memory, shared-disk, or shared-nothing architecture. (Refer to Figure 1.2a–c)	Usually employs shared-nothing architecture. (Refer to Figure 1.2c)

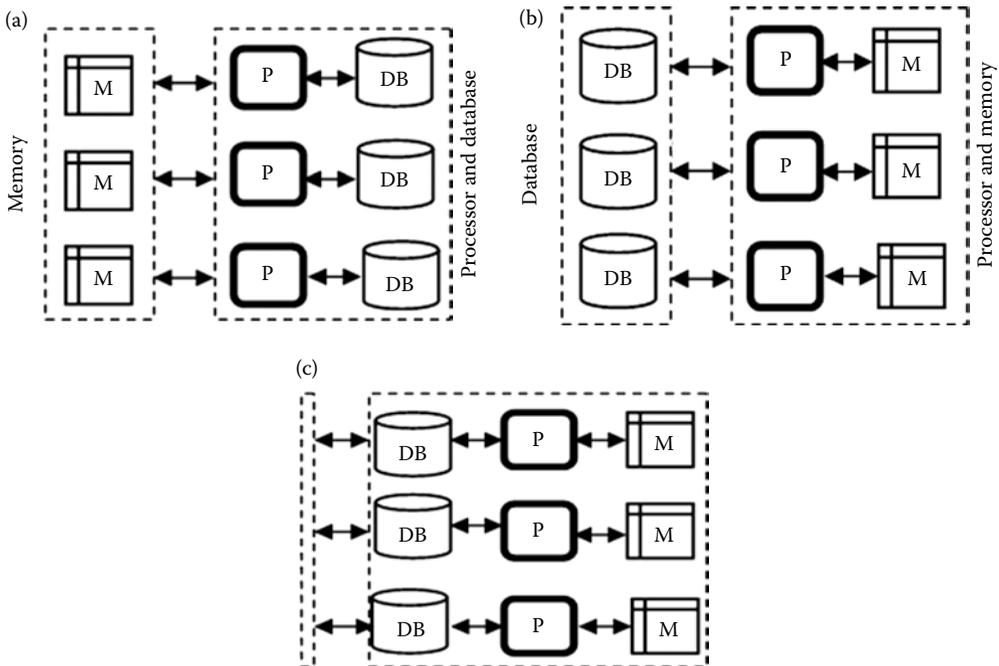


FIGURE 1.2

Different types of architecture for passing information. (a) Shared memory architecture; (b) shared disk architecture; and (c) database, processor, and memory.

performance through parallelism and availability of data through replication mechanism. This enables bearing very large databases with lofty query or transaction loads.

A DDBMS is then defined as the software system that authorizes the management of the distributed database and makes the distribution apparent to the users (Özsu and Valduriez 2011). DDBMS has several advantages such as the following: (a) management of distributed data with different levels of transparency as the physical placement of data at different sites is not known to the user using it; (b) distribution and network transparency as the user need not worry about the operational details of the network, and also enables location transparency as the user has the freedom of issuing commands from any location without affecting its working; (c) replication transparency is another important advantage of DDBMS as it allows to store copies of data at multiple sites using the data allocation technique; (d) increases reliability and availability since DDBMS has multiple nodes, and if one fails, other nodes are available to perform the desired task; (e) improvement in performance as DDBMS fragments the database to keep data closer to where it is needed most, which reduces access and update time drastically; and (f) scalability, that is, easier expansion as it allows new nodes (computer) to be added anytime without making any changes to the underlying configuration.

1.1.3 Distributed Database Techniques

Distributed processing on DBMSs (Manpreet Kaur 2014) is a proficient approach of progressing performance of applications that manipulate outsized dimensions of data. The two major objectives of the devising of distributed databases are to weed out irrelevant and redundant data accessed during the execution of queries and reduce data switching among nodes located at different locations. The key apprehension of distributed database

system design is to fragment the entities in case of object-oriented databases, relations in case of relational database, and further allocation and replication of the fragments in different nodes of the distributed system.

Fragmentation is the technique of dividing a relations of database into a number of pieces, called fragments, which are then distributed and can be stored in various computers located at different locations/sites. It aims to improve reliability, performance, storage capacity, communication, and security. While performing fragmentation, several rules need to be incorporated to maintain no data loss, preserve functional dependencies, and ensure the consistency of data distribution throughout. The various rules are as follows: (a) if database D is decomposed into fragments D_1, D_2, \dots, D_n , each data item that can be found in D must appear in at least one fragment. This rule is necessary to ensure that there is no loss of data during fragmentation so that the consistency of data distribution can be maintained; (b) it must be possible to define a relational operation that will reconstruct database D from fragments D_1, D_2, \dots, D_n , which guarantees the perseverance of functional dependencies; and (c) if a data item X_i appears in fragment D_i , then it should not appear in any other fragment.

The two most widely known types of fragmentation are *vertical* and *horizontal* fragmentation. In vertical fragmentation, some of the columns (attributes) are stored in one computer, and the rests are stored in other computers that comprises a subset of relations that are created by a subset of columns (values of selected columns). In vertical fragmentation, no selection condition is used, whereas in *horizontal fragmentation*, the relations are divided horizontally, that is, some rows (tuples) of the relation are placed in one computer and the rest are placed in other computers and comprises a horizontal subset of relations containing tuples that satisfy selection conditions. A new type of fragmentation introduced is the *mixed fragmentation*, where the techniques of both horizontal and vertical fragmentation are combined. It is also sometimes known as *hybrid fragmentation* (see Figure 1.3a–c).

The placement of data can be done using different strategies of allocation and replication. In *Allocation*, each fragment is stored at the site with optimal distribution. In *Replication*, the DDBMS may maintain a copy/replica of a fragment at several different sites. The various strategies for placement of data are centralized, fragmented, complete replication, and selective replication. *Centralized* consists of single database and DBMS stored at one site with users distributed across the network; *fragmented* partitions the database into disjoint fragments with each fragment assigned to one site; *complete replication* consists in maintaining

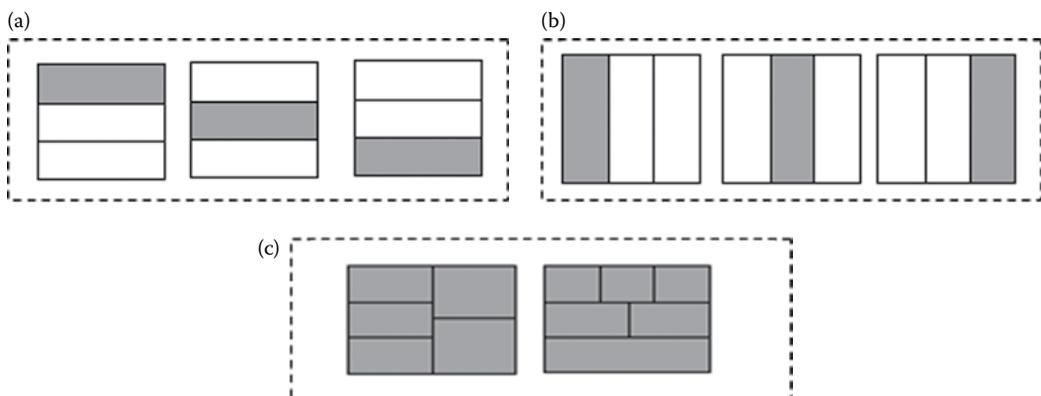


FIGURE 1.3

Different types of fragmentation. (a) Horizontal fragmentation; (b) vertical fragmentation; and (c) hybrid or mixed fragmentation.

a complete copy of the database at each site and is a combination of fragmentation, replication, and centralized. Data are replicated to all sites, which improves their availability and hence performance for retrieval queries, but slows down update operation. *Selective replication* is a combination of fragmentation, replication, and centralized. Some data items are fragmented to achieve high locality of reference, and others that are used at many sites and are not frequently updated are replicated; and others are centralized.

1.1.4 Concurrency Control in Distributed Database

The concurrency control of a system distributed over a computer network is sometimes referred to as *distributed concurrency control*. In database systems and transaction processing, that is, transaction management, distributed concurrency control refers primarily to the concurrency control of a distributed database. In recent years, DBMSs work in multiuser environment where users access the database concurrently. Therefore, the DBMSs control the concurrent execution of user transactions so that the overall correction and update of the database are maintained allowing the user to access database in a multiprogramming approach safeguarding the misapprehension that each user is working single-handedly on a dedicated system. However, to achieve so and to ensure that the update performed by one user does not hamper the task performed by another user are the major concerns to be addressed. For example, consider an online railway reservation system wherein, let us say, two customers X and Y simultaneously try to reserve a seat in the same train. In the absence of concurrency control module in a DDBMS or in a DBMS, the activity would lead to the scenario depicted in [Figure 1.4](#).

In the above situation, although both customers X and Y reserved the same berth (56, Coach A1), the database reflects only one activity, and the other reservation is lost by the system, thereby violating the concept of consistency and durability. Concurrency control in DBMSs is achieved by a program called the scheduler, whose goal is to order the operations of transactions in such a way that the resulting log is serializable. Practically, Two-Phase Locking (2PL) is the most popular scheduling technique for the centralized DBMSs. However, for distributed DBMSs, 2PL induces a high communication charge because of the deadlock predicament. Therefore, an improved algorithm for concurrency control in

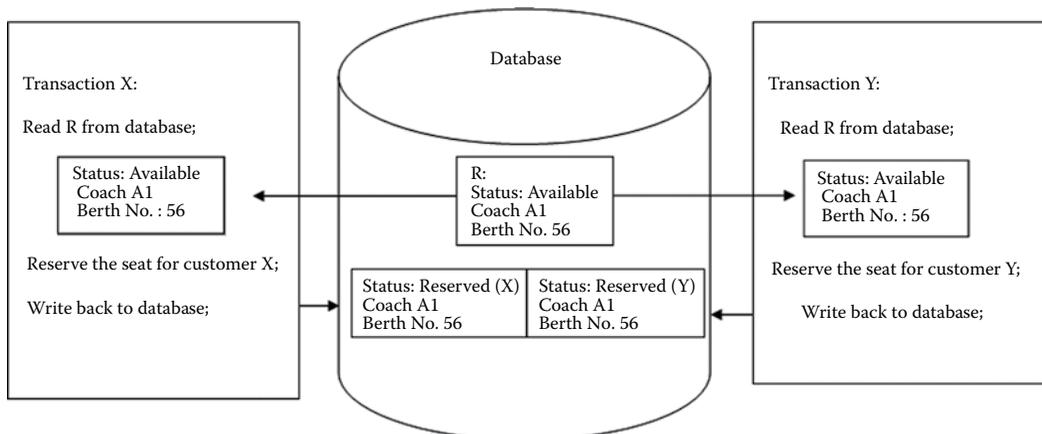


FIGURE 1.4
Scenario of database in the absence of concurrency control module.

DDBMSs is one of the vigorous research areas in database theory. Distributed transaction, like local transactions, must preserve ACID (atomicity, consistency, isolation, and durability) properties; however, it is very intricate to attain the same as the breakdown can occur in any process trying to access the distributed data. Therefore, the ACID property is maintained in distributed transaction processing using (a) recoverable processes and (b) commit protocol. (The details of ACID property shall be discussed in the next section.)

The various algorithms for concurrency control in distributed systems are as follows:

1. *Several extension of 2PL to DDBMS.*

- a. *Centralized 2PL:* In this method, a single site is responsible for the lock management, that is, one lock manager (LM) for the whole DDBMS. Lock requests are issued to the LM. Coordinating transaction manager (TM at the site where the transaction is initiated) can make all locking requests on behalf of local TMs. The communication that occurs in centralized 2PL in distributed environment has been projected in [Figure 1.5](#). *Advantage:* Easy to implement. *Disadvantages:* Bottlenecks and lower reliability; replica control protocol is additionally needed if data are replicated.
- b. *Primary copy 2PL:* In this method, several LMs are distributed to a number of sites and each LM is responsible for managing the locks for a set of data items. For replicated data items, one copy is chosen as primary copy, and others are slave copies. Only the primary copy of a data item that is updated needs to be write-locked. Once primary copy has been updated, the change is propagated to the slaves. *Advantage:* Lower communication costs and better performance than the centralized 2PL. *Disadvantage:* Deadlock handling is more complex in this algorithm.
- c. *Distributed 2PL:* In this method, LMs are distributed to all sites. Each LM is responsible for locks for data at that site. If data are not replicated, it is equivalent to primary copy 2PL; however, if data are replicated, the “read one, write all” (ROWA) replica control protocol is implemented. Read(x): Any copy of a replicated item x can be read by obtaining a read lock on the copy. Write(x): All copies of x must be write-locked before x can be updated. Communication structure of the distributed 2PL: The coordinating TM sends the lock request to the LMs of all participating sites, and the LMs pass the operations to the

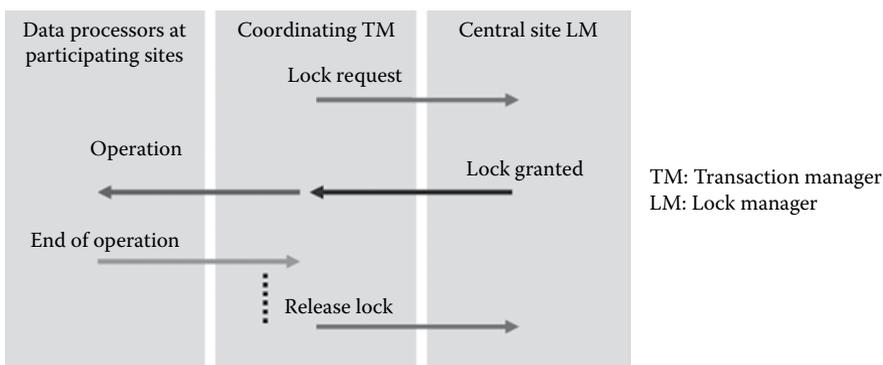


FIGURE 1.5

Communication in centralized 2PL in distributed environment.

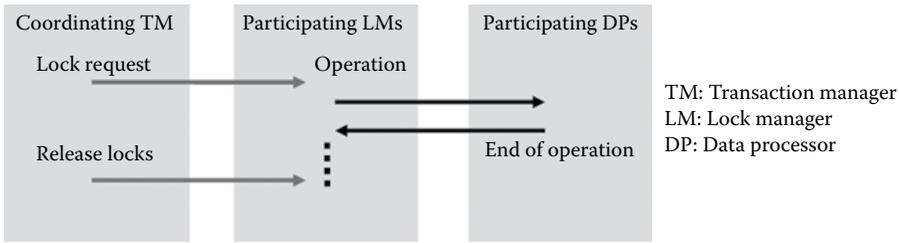


FIGURE 1.6
Communication in distributed 2PL in distributed environment.

data processors. The end of the operation is signaled to the coordinating TM (Armand Wilson 2003). The communication that occurs in distributed 2PL in distributed environment is projected in Figure 1.6. *Advantage:* Better than primary copy 2PL. *Disadvantages:* Deadlock handling is more complex; communication costs are higher than that of primary copy 2PL algorithm.

2. *Wound-Wait (WW):* Also known as distributed wound-wait locking algorithm. This method, like 2PL, follows the “read any, write all” rule. It differs from 2PL in its management of the deadlock problem: Rather than maintaining waits-for information and then confirming for local and global deadlocks, deadlocks are avoided via the utilization of timestamps. Each transaction is numbered according to its initial startup time, and younger transactions are prevented from making older ones wait. If an older transaction requests a lock, and if the request would direct to the older transaction waiting for a younger transaction, the younger transaction is “wounded” and it is restarted unless it is already in the second phase of its commit protocol (in which case, the “wound” is not fatal and is simply ignored). Younger transactions can wait for older transactions. Illustration of wound-wait algorithm is highlighted in Figure 1.7. *Advantages:* The likelihood of deadlocks is eradicated because any cycle of waiting transactions would have to embrace at least one instance where an older transaction is waiting for a younger one which is blocked as well, and this is prevented by the algorithm (Manpreet Kaur 2014).

For example, $t(T1) > t(T2)$: If requesting transaction [t(T1)] is younger than transaction [t(T2)] that has holds lock on requested data item, then requesting transaction [t(T1)] has to wait; *$t(T1) < t(T2)$:* If requesting transaction [t(T1)] is older than transaction [t(T2)] that has holds lock on requested data item, then requesting transaction [t(T1)] has to abort or roll back.

3. *Basic Timestamp Ordering (BTO):* This method, like wound-wait, employs transaction startup timestamps, but it utilizes them differently. Rather than using a locking approach, BTO correlates timestamps with all currently accessed data

	T1 is allowed to
$t(T1) > t(T2)$ →	Wait
$t(T1) < t(T2)$ →	Abort and rollback

FIGURE 1.7
Illustration of wound-wait algorithm.

items and requires that conflicting data accesses by transactions be performed in timestamp order. Transactions that attempt to perform out-of-order accesses are restarted. When a read request is received for an item, it is permitted if the timestamp of the requester exceeds the item's write timestamp. When a write request is received, it is permitted if the requester's timestamp exceeds the read timestamp of the item; in the event that the timestamp of the requester is less than the write timestamp of the item, the update is simply ignored. For replicated data, the "read any, write all" approach is used, so a read request may be sent to any copy, while a write request must be sent to (and approved by) all copies.

4. *Distributed Optimistic (OPT)*: This is a distributed, timestamp-based, optimistic concurrency control algorithm that operates by exchanging certification information during the commit protocol. For each data item, a read timestamp and a write timestamp are maintained. Transactions may read and update data items freely, storing any updates into a local workspace until commit time. For each read, the transaction must remember the version identifier (i.e., write timestamp) associated with the item when it was read. Then, when all of the transaction's cohorts have completed their work and have reported back to the master, the transaction is assigned a globally unique timestamp. This timestamp is sent to each cohort in the "prepare to commit" message, and it is used to locally certify all of its *reads and writes* as follows:

A *read request* is certified if: (a) The version that was read is still the current version of the item, and (b) No write with a newer timestamp has already been locally certified. A *write request* is certified if: (a) No later reads have been certified and subsequently committed, and (b) No later reads have been locally certified already.

1.1.5 Promises of DDBMS

DDBMS is a software system that manages the distributed database and makes this distributor transparent to the user. In DDBMS, "No. of processing elements are known as sites" that are connected over a network. The main objective of DDBMS is to facilitate quick and easy access of data for users who are located at different locations. In DDBMS, data are distributed and accumulated at different sites (single site refers to a single computer). The individual groups can have exclusive local control on the data, therefore making them less reliant on other sites. Every site is considered as a database system of its own and each site is supervised autonomously, and local data are stored on a local computer database and different sites are connected to each other using high-speed network instead of using multiprocessing configuration. The various promises of DDBMS are its functionality/traits that make it finer over DBMS as the DDBMS has complete functionality of a DBMS.

1. *Transparent management of distributed, fragmented, and replicated data*: Transparency refers to separation of higher-level semantics of a system from lower-level implementation details, from data independence in centralized DBMS to fragmentation transparency in DDBMS.
2. *Improved reliability and availability through distributed transactions*: Increases reliability and availability since DDBMS has multiple nodes, and if one fails, other nodes are available to perform the desired task.

3. *Improved performance*: Since each site handles only a portion of a database, the contention for CPU and I/O resources is not that severe. Data localization reduces communication overheads.
4. *Higher system extendibility*: Ability to add new sites, data, and users over time without major restructuring.
5. *Improved performance*: Single-site failure does not affect performance of system.

1.2 Introduction to Distributed Transaction Processing

A distributed transaction is a database transaction (Armand Wilson 2003) in which two or more network hosts are involved. Usually, hosts provide transactional resources, while the TM is responsible for creating and managing a global transaction that encompasses all operations against such resources. Distributed transactions, as any other transactions, must satisfy ACID properties. A distributed transaction is composed of several sub-transactions, each running on a different site.

1.2.1 Background of Distributed Transaction Processing

A DDBMS contains four components: *transactions*, *TM*, *data manager (DM)*, *network concurrency control scheduler (CCS)*, and *data*. *Transactions* communicate with *TMs*, *TMs* communicate with *DMs*, and *DMs* manage the *data*. *TMs* oversee transactions. Each transaction executed in the DDBMS is supervised by a single *TM*, meaning that the transaction issues all of its database operations to that *TM*. Any distributed computation that is needed to execute the transaction is managed by the *TM* (Manpreet Kaur 2014).

Four different operations are identified for the transaction: (a) *READ(X)* returns the value of *X* (a logical data item) in the current logical database state, (b) *WRITE(X, new value)* creates a new logical database state in which *X* has the specified new value, and (c) *BEGIN and END* operations to bracket transaction executions. *DMs* manage the stored database, functioning as backend database processors. In response to commands from transactions, *TMs* issue commands to *DMs* specifying stored data items to be read or written.

Distributed system components engrossed in transactions can take role of (a) transactional client, (b) transactional server, and (c) coordinator. The transaction that takes place in distributed database system is shown in [Figure 1.8](#). *Transactional client* only sees transactions through the transaction coordinator. It invokes services from the coordinator to begin, commit, and abort transactions. Implementation of transactions is transparent for the client and cannot differentiate between server and transactional server. *Transactional server* registers its participation in a transaction with the coordinator and has to implement a transaction protocol (two-phase commit). Every component with a resource accessed or modified under transaction control and transactional server has to know coordinator. *Coordinator* plays key role in managing transaction and is the main component that handles begin/commit/abort transaction calls. Coordinator allocates system-wide unique transaction identifier, and different transactions may have different coordinators.

Each database manager (*DM*) can decide to abort (the veto property). An atomic commitment protocol (*ACP*) is run by each of the *DMs* to ensure that all the sub-transactions

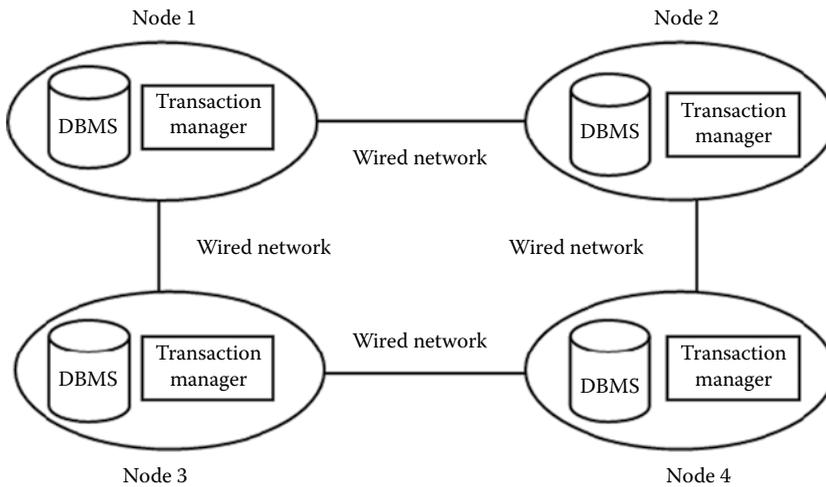


FIGURE 1.8
Transactions in distributed database system.

are consistently committed or aborted. A correct ACP ensures the following: (a) all the DMs that reach a decision, reach the same decision; (b) decisions are not reversible; (c) a commit decision can only be reached if all the DMs voted to commit; (d) if there are no failures and all the DMs voted to commit, the decision will be commit; (e) at any point, if all failures are repaired and no new failures are introduced, then all the DMs eventually reach a decision.

1.2.2 Introduction to Distributed Transaction Processing Models

Many applications involve long transactions, and to deal with such complex applications, much transaction processing systems provide mechanisms for imposing some structure on transactions, which is basically known as transaction processing models.

The distributed transaction processing models are meant for transaction that updates data on two or more computer systems connected via network that must update distributed data avoiding any kind of failure. Thereby, designing and developing a transaction model is complex as there may crop up many types of failures and problems, such as failure at client side, server side failure, connection failure, site failure, network problems, data duplication, distributed deadlocks, distributed transaction at multiple sites, and many more. However, most of the existing techniques are customized for a single transaction point. Furthermore, bonding several distributed points necessitates the existence of high network security technology to sense and improve from any such failure (Ashish Srivastava 2012).

When computers communicate with each other, a TM propels, sets up, commits, and aborts messages to its entire subordinate TMs. It does so as follows: (a) the client begins the transaction by making a request to the TM and deploys the request; (b) the TM on receiving the request from the client processes it by saving the updates; and (c) TM after processing the client request maintains the sequential transaction log, thereby ensuring the durability related to the decision made by it in regard to commit or abort.

Based on the simple concept as discussed, there are several models proposed and available for distributed transaction processing.

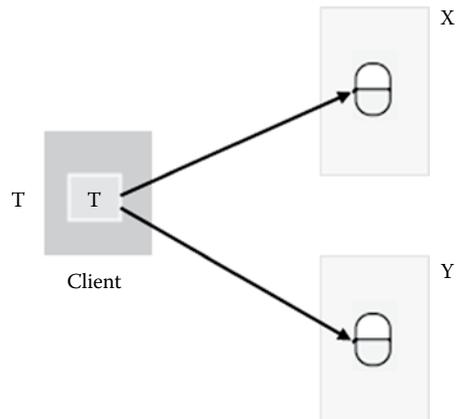


FIGURE 1.9
Structure of flat transactions.

1.2.2.1 Atomic Actions and Flat Transactions

The term atomic action describes a unitary action or object that is essentially indivisible, unchangeable, whole, and irreducible. Flat transaction consists of atomic actions performed on local variables by accessing single DBMS using call or statement level interface. From the client's point of view, the procedure must be accomplished indivisibly. The abort command triggered causes the execution of a program that restores the variables updated by the transaction to the state they had when the transaction first accessed them, thereby disallowing partial rollback and making the total rollback (abort) possible. The structure of flat transaction is highlighted in [Figure 1.9](#).

Several issues related to flat transactions are as follows: limited access to a single DBMS; entire transaction takes place at a single point in time; and at a particular point, if there is some kind of failure, the entire updates/work is lost, that is, if one action fails, the whole transaction must be aborted.

1.2.2.2 Nested Transactions

A nested transaction is a database transaction that is initiated by an instruction within the scope of an already started transaction. A transaction at times may consist of more than one (many) sub-transactions, which further may be composed of several sub-transactions resulting in an arbitrarily deep chain of command/hierarchy of nested transactions. The main or the root transaction, which is not included in any other transaction, is known as the top-level (TL) transaction. Transactions comprising sub-transactions are called parents, and their sub-transactions are their children. The architecture of flat transaction is highlighted in [Figure 1.10](#).

Nested transactions enable committing and aborting the sub-transactions independently of the larger transactions. However, there are several rules to be taken into account while performing nested transactions:

1. While the sub-transaction (child) is active, the parent transaction may not perform any operations other than to commit or abort, or to create more sub-transactions (child).

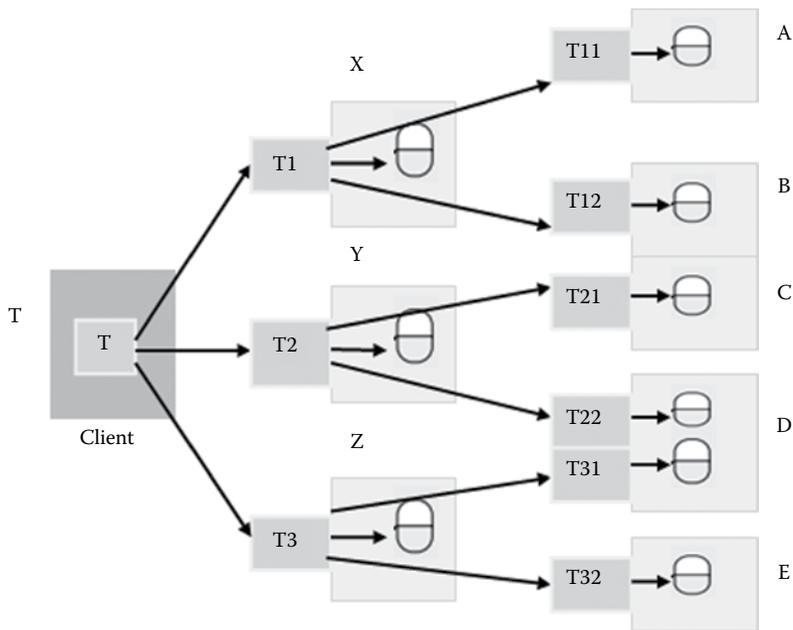


FIGURE 1.10
Structure of nested transactions.

2. The commit operation performed on the sub-transactions has no effect on the state of the parent transaction, that is, parent transactions are still uncommitted. Moreover, the parent can view the modification made by child transaction, but these modifications will be hidden to all other transactions until the parent transaction also commits.
3. If the parent transaction is either aborted or committed while having active child transactions, the child transactions are resolved in the same way as the parent, that is, if the parent aborts, then the child transactions abort, and if the parent commits, then whatever modifications have been performed by the child transactions are also committed.
4. The depth of the nesting with nested transaction is limited only by memory and cannot be restricted by external mechanism.

1.2.3 Distributed Transaction Processing in Relational and Non-Relational Database

Transaction processing basically deals with processing information or data that are divided into individual, indivisible rather atomic operations called transactions. It is designed to maintain the integrity database or file systems in a consistent state, ensuring that mutually dependent operations on the system are either all completed successfully or all terminated successfully (Sheetlani Jitendra 2012). A transaction generally represents any change in database with two main purposes: (a) providing reliable units of work that allow correct recovery from failures and keeping a database consistent even in cases of system failure and (b) providing isolation between programs accessing a database concurrently.

A database transaction, by definition, must be atomic, consistent, isolated, and durable. Database practitioners often refer to these properties of database transactions using the acronym ACID, and ACID properties are considered to be the essential properties of relational DBMS (RDBMS) to ensure successful transaction. (ACID properties for transaction are described in detail in the subsequent sections.)

RDBMSs are compliant with the ACID, but NoSQL (Non-SQL/Not Only SQL) follows a different approach. It follows the CAP theorem laid down by Eric Brewer. This theorem states that it is mathematically impossible for a NoSQL DBMS to comply with all three (C, A, P) features listed down. Therefore, a NoSQL DBMS may only choose two of the features.

The various approaches of how transactions access data in a distributed database in transaction processing are as follows: remote SQL (Structured Query Language) statements, distributed SQL statements, shared SQL for remote and distributed statements, remote transactions, and distributed transactions.

1.2.3.1 Distributed Transaction Processing in Relational Database

A relational database is a digital database whose organization is based on the relational model of data, as proposed by E. F. Codd in 1970. The various software systems used to maintain relational databases are known as a RDBMS. Generally, all RDBMS uses SQL as the language for querying and preserving the database.

Distributed relational database architecture (DRDA) is a database interoperability standard from The Open Group that was first used in DB2 2.3, which describes the basic architecture and rules for accessing the distributed data in distributed relational databases. The structural components, messages to be transmitted, and protocols to be opted in DRDA are defined by distributed data management architecture (DDM). While designing the distributed applications, the designer needs to consider several issues such as data management, security, time factor, amount of data to be transmitted, and frequency of data.

Distributed transaction processing in relational database takes place as follows:

1. *The application requester (AR)* accepts requests in the form of SQL query from an application and upon receiving, sends for processing to the appropriate application servers.
2. *The application server (AS)* on receiving requests from AR processes the section that it can process and forwards the remainder section to database servers for subsequent processing. The communication between the AR and the AS is enabled using the protocol termed as Application Support Protocol, which handles data representation conversion.
3. *The database server (DS)* generally supports distributed requests and forwards parts of the request to collaborating DS in order to fulfill the request. DS on receiving requests from other DS servers or AS processes it, and the communication between DS and AS is established through a protocol called the Database Support Protocol.

The distributed query statements retrieve information from two or more nodes distributed over a network, and to access data from the local database as well as the remote database, distributed SQL query can be initiated. The decomposition of SQL statements is important because it determines the number of records or even tables that must be sent

through the network. Knowledge of how the optimizer decomposes SQL statements can help you achieve optimum performance for distributed queries. If an SQL statement references one or more remote tables, the optimizer must decompose the SQL statement into separate queries to be executed on different databases.

For example, the following query accesses data from the local database “DEPT” as well as the remote “EMP” database as follows:

```
SELECT DNAME, ENAME
FROM DEPT, EMP@REMOTE
WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

is decomposed as

```
SELECT DEPTNO, DNAME FROM DEPT;           which is executed locally
                                           and
SELECT DEPTNO, ENAME FROM EMP;           which is sent to the
                                           remote database.
```

The data from both tables are joined locally. All this is done automatically and transparently for the user or application. Similarly, a distributed update statement modifies data on two or more nodes and is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes.

The database sends statements in the program to the remote nodes, and their execution succeeds or fails as a unit. For example, the following PL/SQL program unit updates tables on the local database and the remote “sales” database:

```
BEGIN
  UPDATE daily.dept@sales.in.india.acme_auto.com
     SET loc = 'NEW DELHI'
     WHERE deptno=26;
  UPDATE daily.emp
     SET deptno=11
     WHERE deptno=26;
END;
COMMIT;
```

1.2.3.2 Distributed Transaction Processing in Non-Relational Database

A non-relational database is a database that does not incorporate the table/key model as that of RDBMS promote; however, it makes use of key–value pairs in documents for data storage. The most popular emerging non-relational database is called NoSQL. The various other family databases that are operated in non-relational manners are Hadoop/Hbase, Accumulo, Hypertable, Amazon SimpleDB, Cloud Data, HPCC, MongoDB, CouchDB, RethinkDB, Terrastore, Cassandra, RaptorDb, and many more. However, MongoDB is a fine example of an NoSQL-based DBMS.

Non-relational database uses aggregate data models that enable usage of data structure to solve several problem domains as modeled by developers (Sheetlani Jitendra 2012). An aggregate is a collection of data (unit) that makes it easier for the database to handle data storage over group when the unit of data resides on any machine. When data are to be retrieved from the database, it helps to get all the related data since the distribution mechanism enhances the movement of aggregate and all the related data are contained

in the aggregate. In that regard, there are two styles of distributing data: (a) sharding and (b) replication.

1. *Sharding* distributes different data across multiple servers, so each server acts as the single source for a subset of data.
2. *Replication* copies data across multiple servers, so each bit of data can be found in multiple places. It can be done in two different ways as follows:
 - 2.1. *Master–slave replication* makes one node the authoritative copy that handles writes, while slaves synchronize with the master and may handle reads.
 - 2.2. *Peer-to-peer replication* allows writes to any node; the nodes coordinate to synchronize their copies of the data.

1.3 Return of ACID Property in Distributed Transaction Processing

Distributed transaction, however, refers to the processing of data located at different locations connected via a network, that is, database systems that focus on distributed transactions as transactions against multiple applications or hosts. In a distributed transaction, a coordinating service ensures that all parts of the transaction are applied to all relevant systems, and if any part of the transaction fails, the entire transaction is rolled back across all affected systems, thereby insisting on the fulfillment of ACID properties over multiple systems or data stores.

1.3.1 Introduction to ACID Property

The acronym for ACID stands for atomicity, consistency, isolation, and durability. This property of database allows sharing of data in a safe and secured fashion avoiding potential inaccuracy and inconsistencies (Sheetlani Jitendra 2012). [Figure 1.11](#) shows the significance of ACID property.

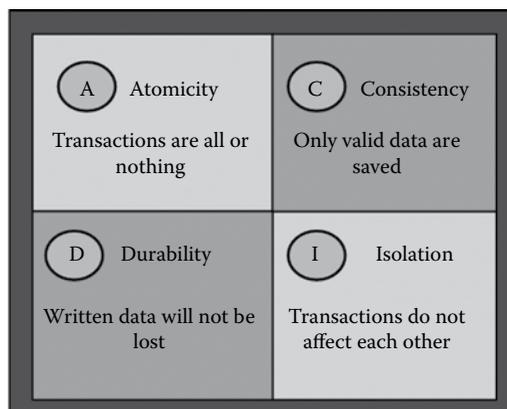


FIGURE 1.11
Significance of ACID property.

Transactions are realized over multiple applications and hosts in distributed database enforcing the ACID properties over multiple data stores making each transaction must follow these four properties.

1. *Atomicity* refers to the ability of the DBMS to guarantee that either all of the jobs of a transaction are performed or none of them and database modifications must follow an “all or nothing” rule. If some part of a transaction fails, then the entire transaction fails, and vice versa.
2. *Consistency* property ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the transaction and after the transaction is over.
3. *Isolation* refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction and helps to retain concurrency of database.
4. *Durability* states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. That means when users are notified of success, the transactions will persist, not be undone, and survive from system failure.

1.3.2 ACID Property and Non-Relational Database

As industries become more and more reliant upon a large quantity of unorganized data such as images, text, files, audio, and videos, traditional RDBMS technologies are proving to be a bottleneck in such kind of situations as they feature a very strict schema. NoSQL is a relatively new DBMS technology for handling large amounts of unstructured data that does not come in a predefined format. Most non-relational databases are incorporated into websites such as Google, Yahoo, Amazon, and Facebook that initiate a slew of new applications with millions of users. RDBMS cannot handle the problem of huge traffic, thereby leading transition toward a new kind of DBMS that is capable of handling Web scale data in a non-relational fashion. The most important feature of a non-relational database is its scalability.

RDBMSs are compliant with the ACID, but NoSQL follows a different approach, that is, CAP theorem laid down by Eric Brewer. This theorem states that it is mathematically impossible for an NoSQL DBMS to guarantee all of the below-mentioned three features. One must always pick two out of the three (C, A, P), that is, CA or CP or AP. The various databases that come under different categories are:

1. *CA category*: RDBMS (MySQL, Postgres), Aster Data, Vertica, and Greenplum.
2. *CP category*: BigTable, HBase, HyperTable, MongoDB, Terrastore, Scalaris, and Redis.
3. *AP category*: Dynamo, Voldemort, CouchDB, SimpleDB, Riak, Tokyo Cabinet, and Cassandra.

The significance of the same is highlighted in [Figure 1.12](#).

1. *Consistency*: All clients view the same instance of data at the same time.
2. *Availability*: Database can be updated, added, or removed without going offline.
3. *Partition tolerance*: Databases can be distributed across multiple servers, and they are tolerant to network failures.

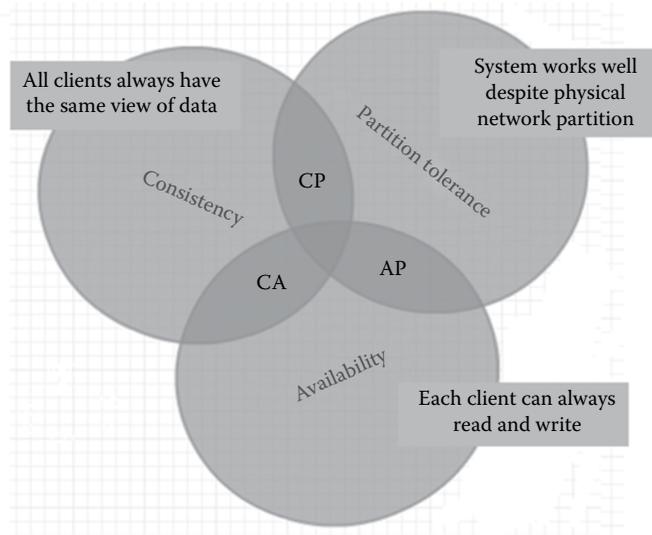


FIGURE 1.12
Significance of CAP theorem.

1.4 NoSQL in Distributed Transaction Processing

NoSQL means Not Only SQL, implying that when designing a software solution or product, there is more than one storage mechanism that could be used based on the needs. NoSQL does not have a rigid definition, but several observations can be framed, such as NoSQL does not employ relational model, open source, schema less, and meant for unstructured data storage and retrieval. There are four types of NoSQL databases: (a) column family stores, (b) key-value pairs, (c) document store, and (d) graph databases.

1. *Column family stores*: Column family databases store data in column families as rows that have many columns associated with a row key (Vatika Sharma 2012). Column families are groups of related data that are often accessed together. Cassandra is one of the popular column family databases; HBase, Hypertable, and Amazon DynamoDB are others (A. Nayak 2013). A typical example of the same is shown in [Figure 1.13](#).
2. *Key-value pairs*: Key-value pair database is one of the simplest NoSQL data stores to use from an Application Programming Interface (API) perspective. The client can get the value for the key, put a value for a key, or delete a key from the data store. Some of the popular key-value databases are Riak (A. Nayak 2013), Redis, Berkeley DB, Amazon DynamoDB, and Couchbase. The example is highlighted in [Figure 1.14](#).
3. *Document store*: Documents are the main concept in document databases. It stores the data in JSON-like documents having key-value pairs. MongoDB is a document database that stores data as a hierarchy of key-value pairs, which allows branching at different levels (a maximum of three levels). Some of the popular document databases are MongoDB, CouchDB, Terrastore, OrientDB, RavenDB, and Lotus Notes that use document storage. Document database such as MongoDB provide

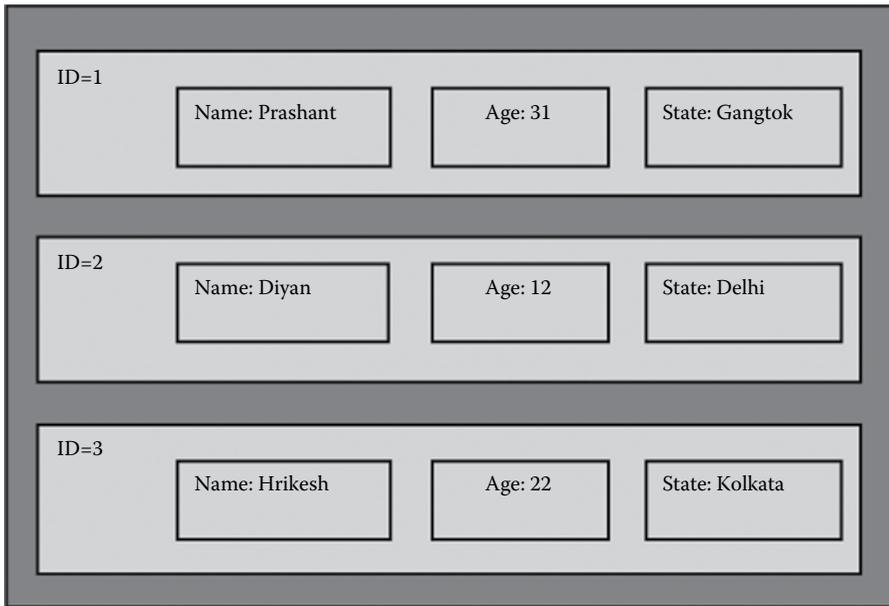


FIGURE 1.13
Column family store database.

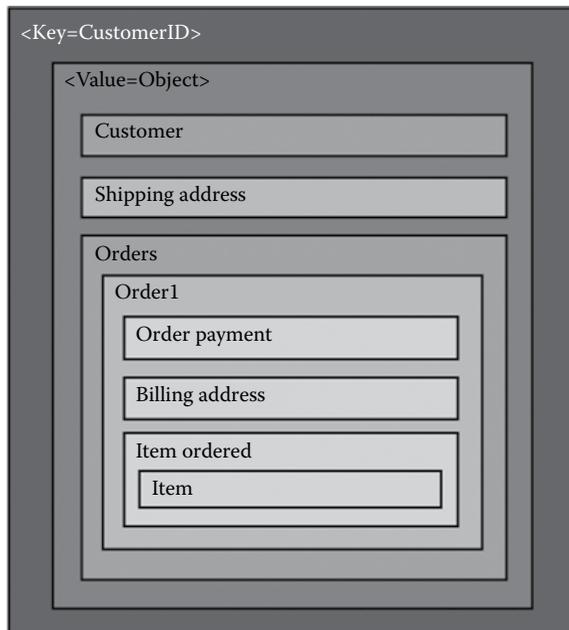


FIGURE 1.14
Key-value pair database.

a rich query language and constructs such as database and indexes allowing for easier transition from relational databases.

Also, MongoDB stores data in JSON-like .BSON files having the structure of the document as given below:

```
{
  "First_name": "Prashant",
  "Middle_name": "Neopaney",
  "Last_name": "Chettri",
  "Age": 31,
  "Address": "{
    "Street": "Namthang",
    "City": "Namchi"
  },
}
```

4. *Graph databases*: These databases store entities and relationships between these entities. Entities are also known as nodes that have properties, and relations are known as edges that can have properties. There are many graph databases such as Neo4J, Infinite Graph, OrientDB, or FlockDB. The graph database scenario taking a typical example of a university employee and the related components is projected in [Figure 1.15](#).

NoSQL has several disadvantages such as the following: (a) no standardized query as that of RDBMS; (b) lacks ACID compliance, which is required in some situations; and (c) no evident increase in performance when used for structured data (Vatika Sharma 2012). So

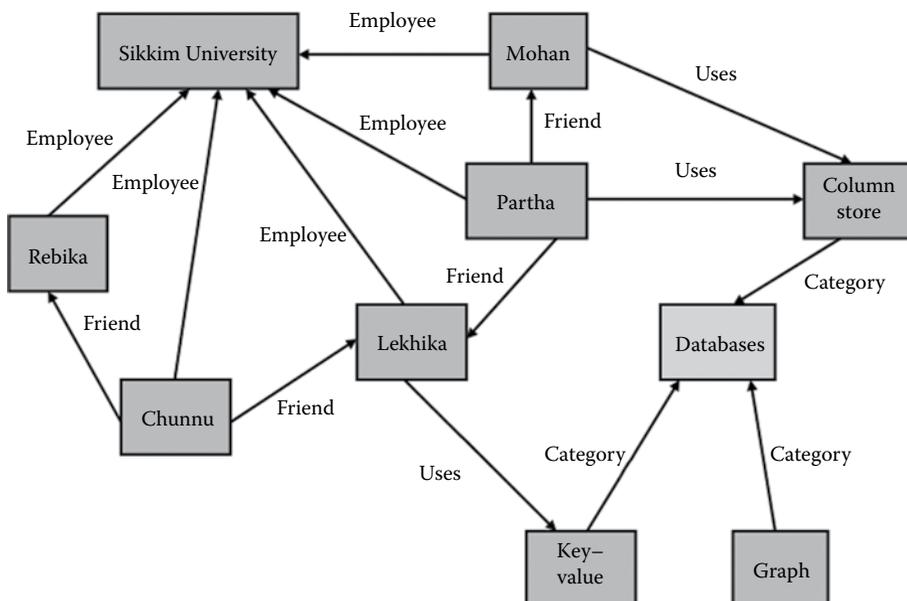


FIGURE 1.15
Graph database.

TABLE 1.2

Comparison of NoSQL and SQL

Basis of Comparison	NoSQL (MongoDB)	SQL (RDBMS)
Data Storage	Stored as key-value pairs in documents.	Stored in a relational model as rows and columns (tables).
Schema Flexibility	Dynamic schema; data can be added, updated, or deleted anytime.	Fixed schema. Altering will result in going offline temporarily.
Specialty	Data which have no definite type or structure.	Data whose type is known in advance.
Scaling	Horizontal: data are stored across multiple servers.	Vertical: more data means bigger servers to handle them.
ACID Compliance	Sacrifice ACID for scalability and performance.	Full compliance with ACID.

TABLE 1.3

Differences in Queries between NoSQL and SQL

Syntax/Query	NoSQL	SQL
To create database	Use db_name	CREATE DATABASE db_name
To drop database	db.dropDatabase()	DROP DATABASE db_name
To insert data	db.db_name.insert({title: "NoSQL"})	INSERT INTO table_name VALUES ("SQL")
To update data	db.db_name.update({'title': 'NoSQL'}, {\$set: {'title': 'NewNoSQL'}})	UPDATE table_name SET title = 'NewNoSQL'

having considered the flaws of NoSQL, which is a popularly used non-relational database, one must wisely choose the DBMS technology, that is, relational or non-relational.

NoSQL differs from RDBMS (G. Harisson 2010; A. Nayak 2013) in the ways that are listed in [Table 1.2](#), and the basic differences between queries of SQL and NoSQL are tabulated in [Table 1.3](#).

1.5 Security Issues in Distributed Transaction Processing Systems

A security constraint consists of a data specification (any subset of transaction processing system) and a security value (given by a classification function). The specific values are unclassified, confidential, secret, and top secret (Lior Okman 2011). Thuraisingham in 1987 defined two types of security constraints: internal and external constraints.

1. *Internal constraints classify* the entire Transaction Processing System (TPS) as well as relations, attributes, and tuples within a relation. These constraints can be applied to data as they are actually stored in the TPS.
2. *External constraints classify* relationships between data and the results obtained by applying operations on the stored data, such as sum, average, and count. Among these constraints are the functional constraints and the dynamic constraints. These security constraints are subject to inconsistency and conflicting local

security constraints. A good global security approach should reject inconsistent security constraints and inconsistent clearance of users. Examples of the *inconsistencies* (Lior Okman 2011; Poonam Dabas 2013) encountered include the following: (a) conflicting security constraints: such constraints classify the same facts into different categories; (b) overlapped security constraints: these constraints cover overlapped data domains; (c) inconsistent security level of replicated data: cases where different copies of replicated data may belong to different security cases; and (d) access privileges of users to replicated data: in stances where a user may have different access rights on replicated data at different sites.

References

- Dabas, P., J. Gurjar, K. Kausal, A. Rawal, 2013. Implementation of security in distributed transaction system—A comparative study. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(7), 1487–1493.
- Harrison, G. 2010. *10 Things You Should Know About NoSQL Databases*. <http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/> Accessed on 7 April 2016.
- Jitendra, S., V.K. Gupta, 2012. Concurrency issues of distributed advance transaction process. *Research Journal of Recent Sciences*.
- Kaur, M. 2014. Transaction processing in distributed databases. *Transaction Processing in Databases*.
- Nayak, A., A. Poriya, D. Poojary, 2013. Type of NoSQL databases and its comparison with relational databases. *International Journal of Computer Science*.
- Okman, L., N. Gal-Oz, J. Abramov, 2011. Security issues in NoSQL databases. *International Joint Conference of IEEE*. Changsha, China, pp. 541–547.
- Özsu, M. T., P. Valduriez, 2011. *Principles of Distributed Database Systems*. Third Edition ISBN: 978-1-4419-8833-1 (Print), 978-1-4419-8834-8 (Online).
- Sharma, V., M. Dave, 2012. SQL and NoSQL databases. *International Journal of Advanced Research in Computer Science and Software Engineering*.
- Srivastava, A., U. Shankar, S. K. Tiwari, 2012. Transaction management in homogenous distributed real-time replicated database systems. *International Journal of Advanced Research in Computer Science and Software Engineering*.
- Wilson, A, 2003. Distributed Transactions and Two Phase Commit, SAP White paper, <http://www.sdn.sap.com/> Accessed on 9 April 2016.

2

X/Open Distributed Transaction Processing Model Using EJB and MTS

S. Gunasekaran and V. Bargavi

CONTENTS

2.1	Introduction	23
2.1.1	Commit Request Phase	24
2.1.2	Commit Phase	24
2.1.2.1	Success	24
2.1.2.2	Failure	24
2.2	Technologies Used in DTP	25
2.2.1	Enterprise Java Bean.....	25
2.2.1.1	Implementing EJB in NoSQL.....	27
2.2.1.2	EJB Integrates Web Services	28
2.2.1.3	NoSQL Integration Using Hadoop.....	29
2.2.2	Microsoft Transaction Server Model.....	30
2.3	Distributed Transaction Support in EJB and MTS.....	31
2.3.1	Distributed Transaction Management in J2EE	31
2.3.2	Transactional Model in EJB.....	31
2.3.2.1	J2EE Transaction in EJB.....	33
2.3.3	Transactional Model in MTS.....	34
2.3.3.1	J2EE Transaction in MTS.....	36
2.4	Comparison between EJB and MTS.....	38
2.5	Indexed Services of XA Interface Used in DTP.....	38
2.6	Research Directions: NoSQL DB for Repository Design for IoT	45
	References.....	46

2.1 Introduction

The distributed transaction is a database transaction where the transaction manager (TM) is responsible for managing and creating a global transaction that encompasses all operations against such resources. All the distributed transactions guarantee that database transaction is processed reliably using ACID properties which stands for atomicity, consistency, isolation, and durability (Bernstein and Newcomer 2009).

The transactions are categorized into synchronous and asynchronous modes. Synchronous transactions are having the timeout mechanism which is monitored by TM to determine the success and failure of transactions. Asynchronous transactions occur in separate sessions by making calls to the resource manager by not depending on the success or failure of the previous one. It uses a trigger mechanism or a polling mechanism to check

whether the resource managers have responded. If each distributed transaction processing (DTP) component is subject to transaction, control will be able to undo its work during a transaction that is rolled back at any time.

The distributed transaction uses a two-phase commit (2PC) algorithm for coordinating the process in a distributed atomic transaction, whether to commit or roll back (Gottlob and Vardi 1995). This protocol is widely used because of its ability to recover from the failure which is achieved using logging of protocol state. The algorithm has two phases:

1. Commit request phase
2. Commit phase

Consider one node as a coordinator is declared as master site and the rest of the nodes are considered as cohorts. At each node, protocol assumes stable storage with a write-ahead log, where no node crashes forever and any two nodes can communicate with each other. The coordinator initiated the protocol after the last step of transaction has been reached. The cohort responds with an agreement or an abort message depending on whether the transaction has been preceding success or not. The process is shown in [Figure 2.1](#).

2.1.1 Commit Request Phase

1. The coordinator sends request to all cohorts to commit message, and then it waits to receive a reply from all cohorts.
2. The cohorts execute their transaction until the point they ask to commit, and each cohort writes an entry to their undo and redo logs.
3. If cohort action is succeeding, then it sends a reply message (votes yes to commit) or it will send an abort message (votes no).

2.1.2 Commit Phase

2.1.2.1 Success

If the coordinator receives a message from all cohorts from the commit request phase:

1. The coordinator sends a commit message to all cohorts.
2. Each cohort completes its operation and releases all its locks, and then resources are held during the transaction.
3. Each cohort will send an acknowledgment to the coordinator, and the coordinator completes its transaction when all acknowledgments have been received.

2.1.2.2 Failure

If any cohort commits No during the commit request phase:

1. The coordinator will send a rollback message to all cohorts.
2. Using undo log, each cohort undoes the transaction and releases the resources, and locks are held during the transaction.
3. The cohort sends an acknowledgment to the coordinator.
4. Then coordinator undoes the transaction when all the acknowledgments have been received.

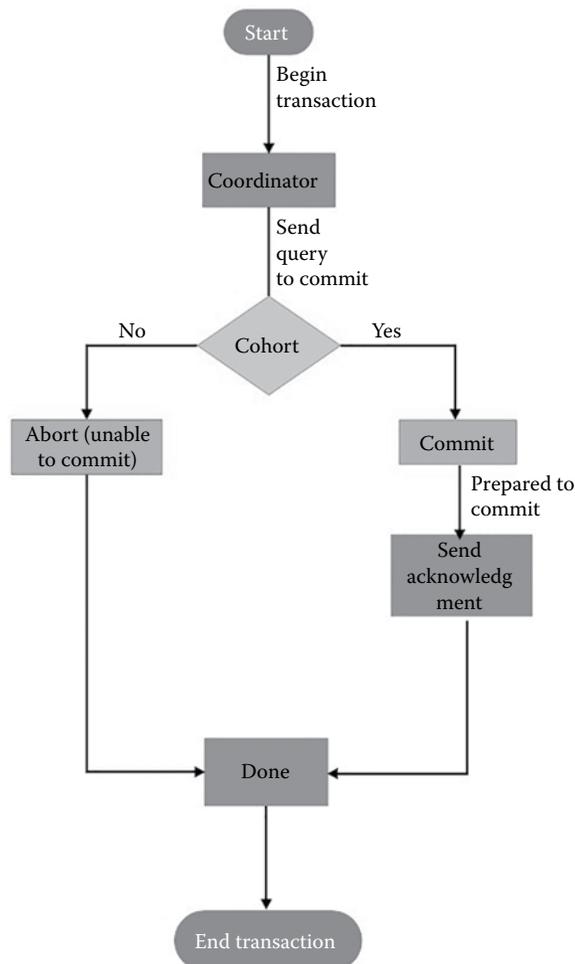


FIGURE 2.1
Two-phase commit protocol.

2.2 Technologies Used in DTP

There are several technologies that support distributed transaction standards. Among them, Enterprise Java Bean (EJB) and Microsoft Transaction Server (MTS) are the most predominant technologies that fully support transaction standards (Singh and Huhns 2005). The overview details of EJB and MTS are as follows.

2.2.1 Enterprise Java Bean

It is a server-side segment, which embodies the business rationale of an application and a subset of the Java EE particular. EJB is a particular for making versatile, value-based, multiclient, server-side, and secure endeavor-level application. It characterizes a reliable segment design system for making multitier middleware and set of detail. The EJB of a

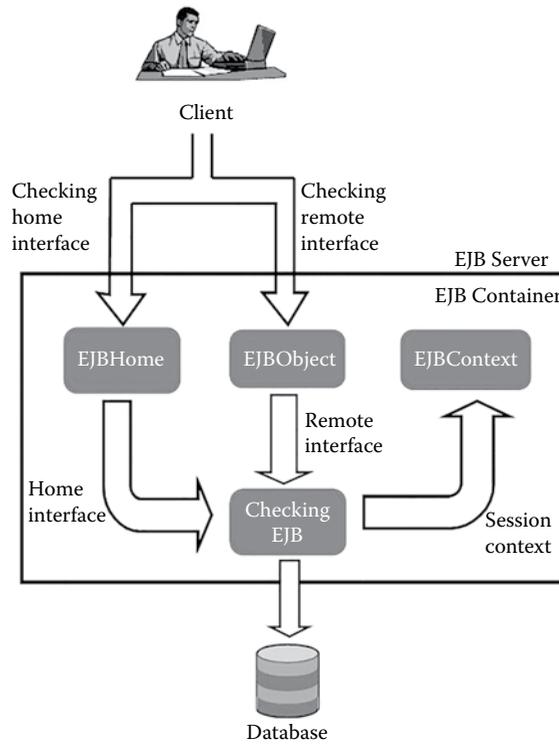


FIGURE 2.2
EJB architecture.

web compartment gives a run-time environment to web-related programming segments, including exchange handling, Java servlet life cycle administration, PC security, and web administrations. [Figure 2.2](#) explains the basic architecture of EJB model. It consists of

- An EJB Server
- Remote EJBObjects, Home Objects, and Enterprise Bean that run within the container
- EJB Client
- EJB Container that runs within the server

The Java part which keeps running under the control of EJB holder is called EJB components. This part is created with detail and is executed as one or more EJB components. EJB parts are sent to EJB server and keep running in a holder which oversees them. The article executes `javax.ejb.EJBHome` interface which serves as a Factory Object to discover existing parts in the event of substance beans and to make a new occasion of these components.

The EJB insert a Remote Object is also known as EJBObject between an actual EJB component that the EJB container manages and its client, and a Home object is called as a Factory Wrapper Object (Sriganesh et al. 2006). The client can make a call to an EJB component through Wrappers (Home and EJBObject), which intercept the call and make their own instance management algorithm called Instance Pooling, and then the Wrapper passes the call to the actual EJB component as shown in [Figure 2.2](#).

The EJB component makes a call to the context object through Entity context or Session context interface. It handles component on the container, where component gets security information, transaction information, and also information from component deployment descriptor. In every EJB component, there exists a context object that implements the Java. `ejb. SessionContext` or Java. `ejb. EntityContext` interface.

2.2.1.1 Implementing EJB in NoSQL

The EJB can be implemented in NoSQL using Ubuntu as shown in [Figure 2.3](#). The Web service is a stateless session bean, so it is easy to expose as an EJB component. To do this,

- For stateless EJB component has to create web service by changing the EJB deployment descriptor as `ejb-jar.xml`.
- Define a web service using web service description language and deployment descriptor defined as `webservices.xml`.
- Deploy EJB component and Web Services Description Language (WSDL) in an application.

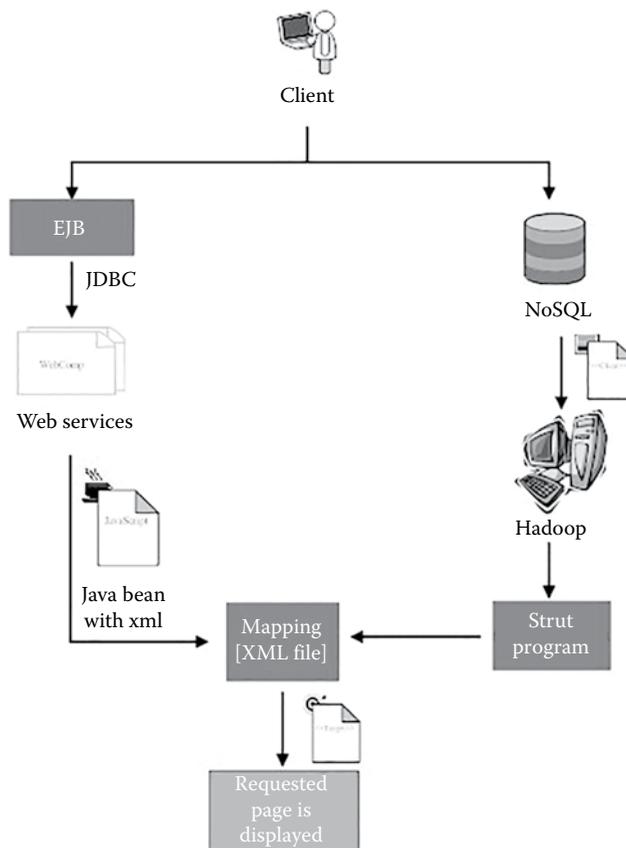


FIGURE 2.3
EJB in NoSQL.

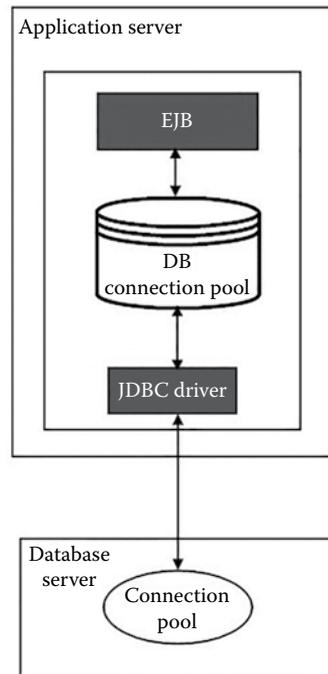


FIGURE 2.4
Connect JDBC using connection pool.

2.2.1.2 EJB Integrates Web Services

The EJB integrates web services using JDBC driver, that is, it is a client/server JDBC driver, server implements necessary part of JDBC interface as web services, and client uses the application by invoking those web services.

To access database resources, JDBC driver uses connection pooling technique, which allows multiple clients to share a set of connection objects. These connection objects are provided to access database resource. [Figure 2.4](#) explains the releasing and the acquiring database connection through database manager, using JDBC where the performance impacts the EJB layer.

To establish a database connection, the application server will take a few minutes, which is needed for each client request, and the connection process includes authentication, communication with the server, and so on.

In [Figure 2.4](#), a JDBC connection established using connection pooling technique is shown. Il-Chul Yoon et al. (2000) state that when the client sends a request, it allows multiple-connection object to share database from the resource pool. During that situation, the J2EE component uses the connection object without collapsing of database in resource manager. The application server implements a database pool in its memory space and optimizes its resource by dynamically altering the pool size. The following code explains the JDBC connection pool in the database:

```

public product ejbCreate() // implement JDBC driver
{
  try
  {

```

```

// Initialize JNDI lookup
Context Ctxt = new InitialContext (parms);
ConnectionpoolDS CDS = (connectionpooldatasource) Ctxt.
lookup(cpresource);
CDS.setDatabaseName ("Pdb");
CDS.setusrIF("xyz");
PooledConnection pc = CDS.getPooledConnection ();
Connection con = pc.getConnection();
Con.close();
}}

```

The connection pool technique does not provide optimum performance for EJB layer. To obtain this optimum performance, the following code explains about the EJB client that accesses an LDAP directory which implements a connection pool:

```

import netscape.ldap.util.*;
public class newCustBean implements SessionBean
{
private SessionCtxt ctxt;
private LDAPConnection LC;
public void setSessionCtxt (SessionCtxt sc)
{
this.context = sc;
//Initialize JNDI lookup parameters
Context ctxt = new InitialContext(parms);
ConnectionPool cp = (ConnectionPool)ctxt.lookup(cpresource);
//LDAP connection is established
try
{
this.lc = cp.getConnection ();}}}}

```

After establishing the JDBC connection, the server side is implemented as servlets and uses Simple Object Access Protocol (SOAP) framework to easily access the web services. The web services have two technologies associated with them as follows: SOAP and WSDL. WSDL acts as an Interface Description Language (IDL), which is adapted to the internet, where an application can publish an API to another application to use the services through WSDL, and SOAP acts as transport protocol.

When a client sends request as an XML document, that is sent to web services as WSDL document. This document describes about the input, its operations, and output. It also includes the URL, which is used to generate client stub class through WSDL document, and it contains a relevant method of operation for web services API. The client can easily integrate using this method to make use of web service operations. The compiled classes are available to servlet engine to run web services by integrating them into servlet engines for startup transaction. After the transaction is completed, the XML file is sent to mapping component.

2.2.1.3 NoSQL Integration Using Hadoop

NoSQL is a distributed architecture which is used to store and retrieve data. After receiving requests from clients, NoSQL integrates with Hadoop for the transaction. The Hadoop generates the XML file using struts. The Strut has three components as follows: action, result, and servletfilter. The servletfilter acts as controller, where each incoming request is inspected to determine how to handle the request. All URL requests are needed to map the

action with XML-based configuration or Java annotations. And then, interceptor executes after and before the request is processed and also provides crosscutting task which can be easily reused and also separate task from other architectural concern. Finally, the action takes place, and it has two important roles in handling the transaction. First, it transfers JSP data from the request to the view, and then, it determines how the framework displays result in response to the request. Now, the XML documents are mapped based on the query sent by the client.

2.2.2 Microsoft Transaction Server Model

Figure 2.5 explains the basic architecture of MTS model. It consists of

- MTS Clients
- MTS server component
- Context Wrapper and Factory Wrapper for each component
- MTS executive

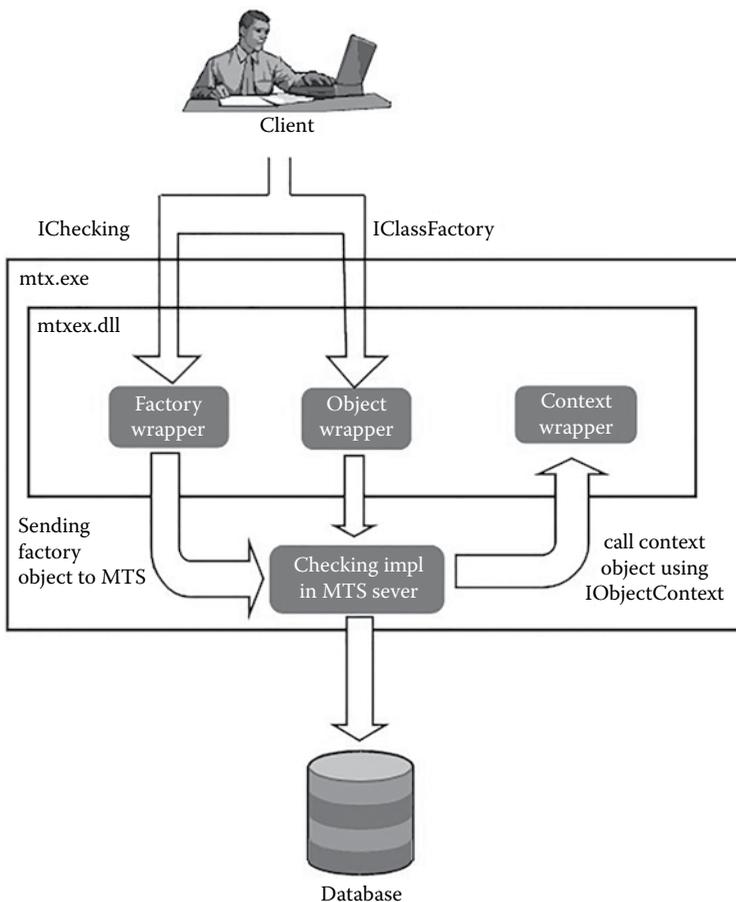


FIGURE 2.5
MTS architecture.

The MTS components are developed as in-proc DLLs and are executed as one or more COM components, and then the components are deployed, which run in the MTS executive that manages them. These COM components run under the control of MTS executive are called MTS components. Whenever an object implements IClassFactory, it serves as a Factory Object to create a new instance for these components. A Factory Wrapper Object and an Object Wrapper are inserted between the MTS components that MTS manages, and its client (Gray and Lievano 1997).

When the client makes a call to MTS component, the wrappers (object and factory) intercept the call and create their own instance algorithm called Just In Time Activation (JITA) into the call, and then the wrapper passes the call to the actual MTS component. In every MTS component, there exists a context object that implements the IObjectContext interface. This component makes a call into the context object through IObjectContext interface, and then the context object maintains specific information about the component such as deployment information, security information, and transactional information.

2.3 Distributed Transaction Support in EJB and MTS

2.3.1 Distributed Transaction Management in J2EE

The transaction treats a sequence of operations as unit for ensuring database integrity and to satisfying requests from client. In J2EE, the transaction has responsibility for transaction management to control dissociation and association of transaction context in which they are executed. It supports both distributed and local transactions. By using two-phase commit protocol, it provides flawless execution, and there is no data loss during the transaction. The process of distributed transaction in both EJB and MTS provides scalability, and declarative transaction management is explained in the following chapter.

2.3.2 Transactional Model in EJB

EJB uses the Java Transaction API (JTA) and Java Transaction Service (JTS) for transactional support. The clients and components can see the exception, and the interface provided by JTA and JTS is not a transaction service, but it provides an interface for the underlying transaction service.

JTS has several exceptions and one service. The EJB transaction model is similar to OTS, for clear understanding look at a list of exceptions, which was modeled to work with OTS, and it can also be used as an interface to other transaction service. The components involved in OTS transaction are as follows:

1. Terminator
2. Control
3. Resource
4. Coordinator
5. Synchronization

All these components are directly applied to EJB transaction support, and [Figure 2.6](#) explains the transaction process in EJB where all objects are participating in the transaction. The Rollback and Commit is applied to all resource objects.

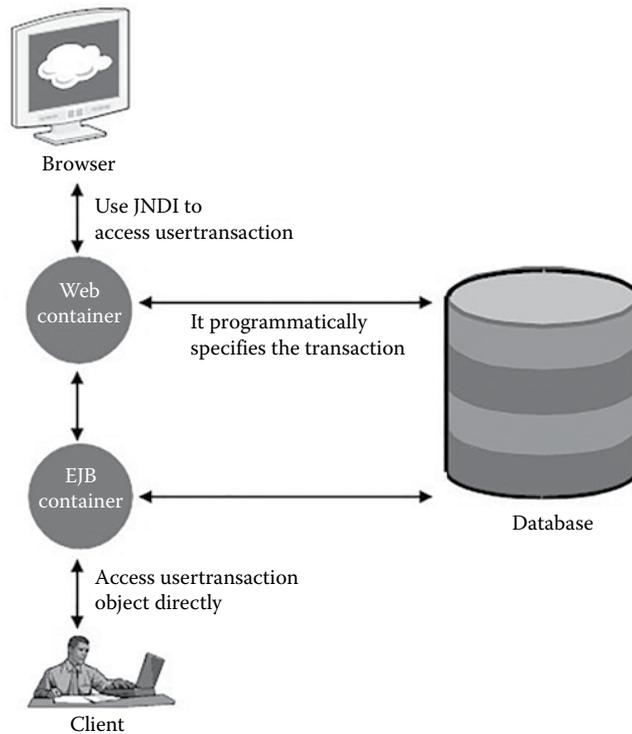


FIGURE 2.6
J2EE transaction in EJB.

The Control Object represents a transaction process, and to manage transaction on behalf of the bean is done by using EJB container and is transparent to EJB developer. The coordinator and terminator are obtained from this transaction as shown in [Figure 2.6](#).

The Resource Object represents a connection to the database, and it has a transaction state. It calls the `commit_()` operation which will force the updates to apply to the database and when a transaction starts a `rollback_()` will revert all changes to the database. When the `rollback_()` or `commit_()` id is done, it unlocks the corresponding rows in the database and then participates in two-phase commit protocol, and it decides which transaction has to get participate either commit or rollback.

The Synchronization Object notifies the completion of transaction to decide whether the transaction was committed or rolled back. In two-phase commit protocol, the object is a passive participant, and it will not decide which commit or rollback transaction is participating.

The Terminator Object is used by the EJB container to commit or roll back a transaction when a thread returns from a bean method. All objects in the transaction will apply appropriate commit or rollback operation when a commit or rollback is requested.

The Synchronization and Resource Object is registered into a transaction using coordinator object, and beans will not directly access the object. The EJB uses transaction-aware objects to obtain a reference to the current transaction coordinator to register them automatically.

The below example explains the use of JTA method to perform a distributed transaction. This code represents a transaction application and a TM. There are two connections for two different data sources to do SQL work below a single transaction.

2.3.2.1 J2EE Transaction in EJB

In J2EE, the DTP model is implemented using JTA where the TM is illustrated by `javax.transaction.UserTransaction`. The JTA that implements transaction ACID properties between multiple resource manager and JTA TM is executed by EJB container, and it is registered in Java Naming and Directory Interface (JNDI) namespace. The client generates a transaction context by looking up a JNDI namespace for naming resources `UserTransaction` that act as a factory method for distributed transactions. The `UserTransaction` interface extracts all classes and interfaces for managing J2EE-distributed transactions. The application developer uses `begin()`, `commit()`, and `rollback()` method of `UserTransaction` context.

It is an n-tier application model for enterprise applications. Based on J2EE, EJB consists of business logic of the application, and its modular server component consists of many features that include improving declarative transaction management and application scalability. The transaction in J2EE takes place in three common J2EE configurations as shown in [Figure 2.6](#). They are

Case 1: Client ↔ EJB container ↔ EIS resources

Case 2: Browser ↔ Web container ↔ EIS resources

Case 3: Browser ↔ Web container ↔ EJB container ↔ EIS resources

Web Container: It acts as an interface between web server and web component. The web component is declared as a JSP page or a servlet. These components are used to create HTML pages which are an application interface and also generate an XML page or other data format. The web components are executed on a web server and make a response to HTTP request from clients ().

EJB Container: The EJB Container provides an environment that supports the execution of an application developed using an EJB. It acts as an interface between J2EE and enterprise bean which allocates business logic in JavaEE application. It runs on the Java EE server and provides the execution of an application enterprise bean.

Case 1: Transaction in Clients—In this transaction, clients can directly access the `UserTransaction` object which entirely depends on the capabilities provided by the container. Transaction support in clients does not require J2EE platform, but it is chosen to provide and implement this capability for added value to ensure portability.

Case 2: Transaction in Web Components—The transaction in web components takes place using JNDI to hold `UserTransaction` object and to specify transaction programmatically using the interface. To specify transaction within web components using JTA interface is illustrated below:

```
ContextObject myContxtObj = new InitialContextObject ();
UserTransac ut = (UserTransac) myContxtObj.lookup("java: compile/
UserTransac");
ut.begin();
//It will perform the transaction
ut.commit ();
```

Case 3: Transaction in Application Server—The following scenario explains the use of transaction on application server built in transaction monitor and EJB container. They are as follows.

Scenario A: The application can easily send or receive messages using the EJB server from one or more JMS destination, or it updates the data in one or more databases using

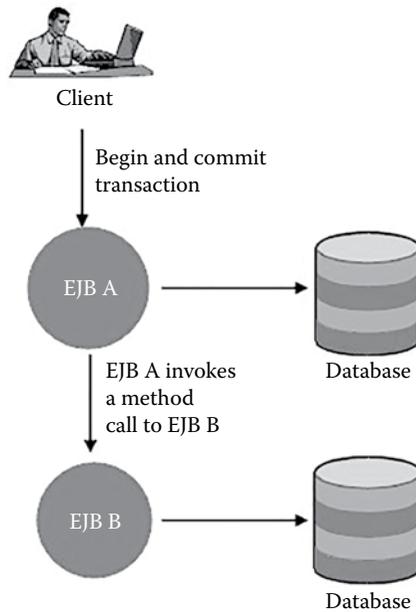


FIGURE 2.7
Scenario A Message send/receive over multiple database.

single transaction. Scenario A (Figure 2.7) explains how the client invokes a method on remote interface. The client can send a method to or receive a method from JMS queue by invoking remote methods of EJB A and update data in database A, and after that it invokes EJB B to update database. When it commits the transaction database system, application server and messaging will perform two-phase commit protocol for ensuring automatic updates for all three resources. It uses standard JMS and JDBC API to update and send messages, and then application server will enlist the session for connection to JMS provider and transaction as a part of the database connection.

Scenario B: In this scenario, a client can access multiple databases through multiple servers in a single transaction. First, it invokes the enterprise bean A which in turn sends message to message queue and then updates the data in the database. Then, EJB A calls a method on enterprise bean B which is deployed on another application server to update the data in the database as shown in Scenario B (Figure 2.8). When EJB A invokes a method on EJB B, both application servers cooperate to propagate transaction context from A to B (Figure 2.9).

2.3.3 Transactional Model in MTS

In MTS model, each component declares transactional primitives, and then automatically runs and initiates environment transactions. It creates a context object for each component present in a transaction, which contains information such as a current activity identifier, security, id of the caller, and transaction identity. In MTS executive (mtxex.dll), the context object is loaded, and in addition, it also takes part in the transaction processing on behalf of the component. These components and MTS executive will execute in a separate process

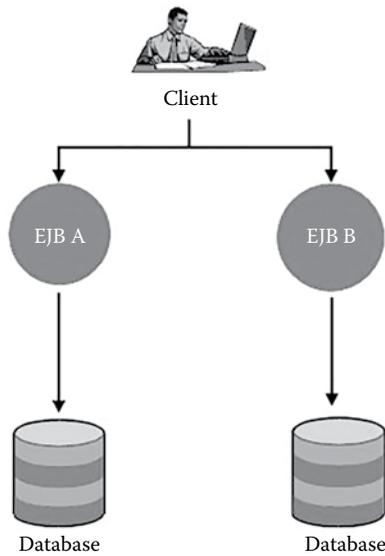


FIGURE 2.8
Scenario B Database updates are updated in same transaction.

given by MTS (mtx.exe). The two-phase commit transaction is carried out by OLE transaction technology. There are three transaction processing entities present in this model:

- TM—it specifies MS DTC.
- Resource Manager—it allows transactional access to the underlying data (e.g., SQL Server) and manages persistent, durable resources such as a file or database.
- Transaction Client.

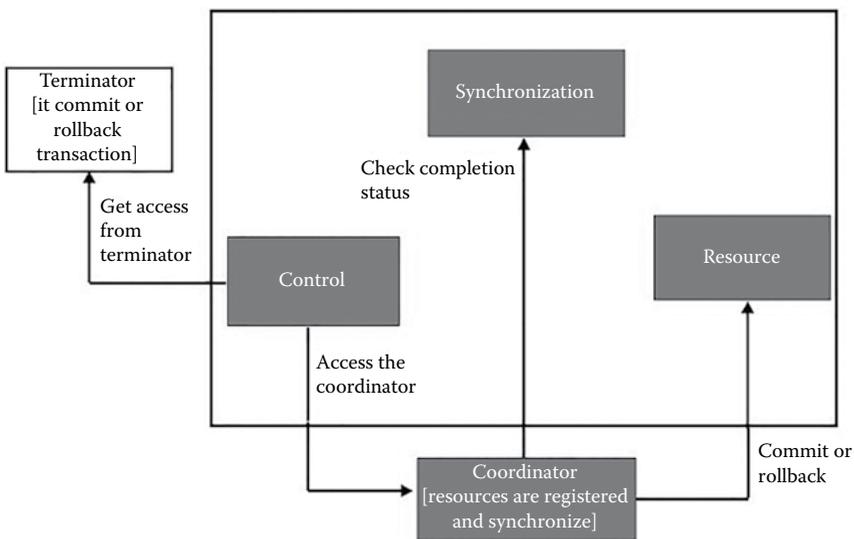


FIGURE 2.9
Transaction in EJB.

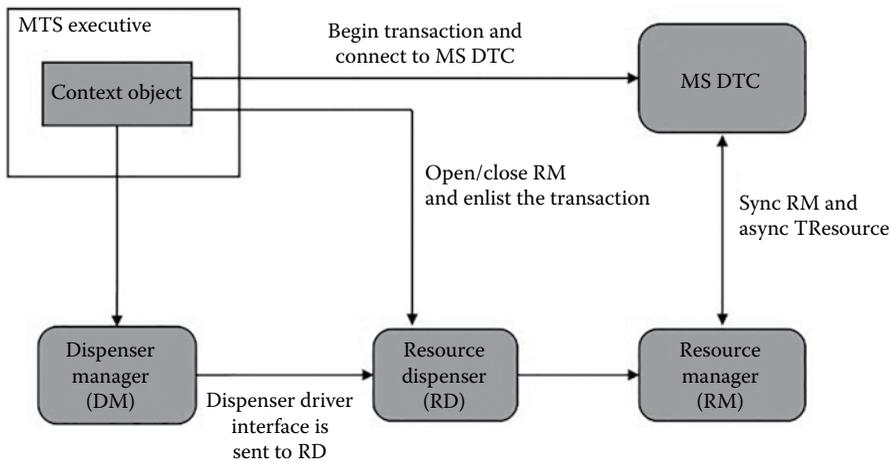


FIGURE 2.10
Transaction in MTS.

Figure 2.10 explains the transactional process in MTS. From each MTS component, it isolates the information, that is, how the transaction is carried out. MTS Executive, Dispenser Manager, and MS DTC will do the work that required to perform two-phase commit transaction. To carry out this process, MTS takes all responsibilities of Transaction Client and performs the following steps:

- Creates and maintains the transaction context.
- Instructs MS DTC to commit or abort transaction that occurs currently.
- Propagates transaction context to all participating objects and enlists the appropriate resources.

MS DTC:

DTC is designed to provide services for ensuring complete and success transaction whenever the system or process fails. It uses two-phase commit protocol:

- Phase 1 includes TM to send request to each enlisted component and prepare to commit the transaction.
- In Phase 2, if all components are successfully prepared, then the TM commits the transaction.

It performs the transaction coordinator's role for each component involved and serves as a TM to each computer which manages the transaction. It performs the following steps:

- To begin a transaction, the application program calls TM.
- When the application program is prepared to commit, it sends wait for the TM decide whether to commit or abort a transaction.

2.3.3.1 J2EE Transaction in MTS

The transaction in J2EE takes place automatically by allowing developers to handle transaction application automatically as shown in Figure 2.11. In J2EE, an application component

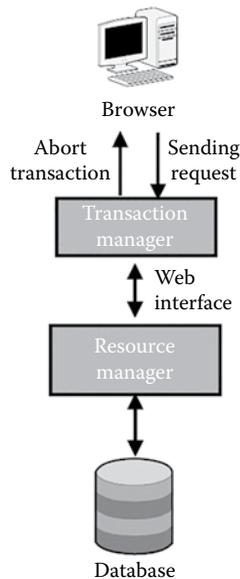


FIGURE 2.11
J2EE transaction in MTS.

is deployed in a container provided by application server where application component requires a transaction to access multiple resource managers and use TM for managing transaction across multiple resource managers.

Transaction Manager:

When the client sends a request, TM handles the transaction. The TM provides JTA to handle the interface between TM and participants involved in a distributed transaction. When the transaction starts, it creates a transaction object to represent transaction. Then, the transaction initiates to make a call to resource manager, and activities of resource manager are tracked during transaction. If any problem occurs during a transaction, TM will abort the transaction, or if the transaction commits, then DTC updates the changes to the database.

Resource Manager:

1. When the application program accepts the transaction, then the TM initiates the two-phase commit protocol. First TM enlists each resource manager to commit the transaction, and the resource manager is ready to commit the transaction. It records all new and old data in a stable storage to recover data whenever the system fails.
2. If a resource manager cannot commit the transaction, then it informs to TM and transaction is aborted, or if it is ready to participate in a transaction, then it informs the TM to either commit or abort transaction. It supports two types of transaction as follows:
 - a. JTA/XA transaction: It controls and coordinates a transaction which is external to resource manager.
 - b. RM/Local transaction: It controls and coordinates a transaction which is internal to resource manager.

3. If a resource manager fails, then all of its enlist transaction is aborted. It recovers from failure, and it sends queries to the TM about the enlisted transaction. The TM informs the resource manager about those transactions. Whenever a resource manager fails, DTC makes the resource manager to do transaction durable and atomic.

2.4 Comparison between EJB and MTS

Description	EJB	MTS
Client-managed transaction	TX_MANDATORY—caller must start transaction, and scope of the client is always invoked in the bean. The client does not have the javax.transaction.Transaction-Required Exception will arise.	The client can have option to explicitly build when a transaction should begin and end. It is automatically built on top of automatic transaction.
Component-managed transaction	TX_BEAN_MANAGED—enterprise bean starts and ends a transaction. It uses javax.transaction.UserTransaction.	The MTS component starts and ends transaction. It directly calls to MS DTC by using OLE transaction.
Component types	There are two major types: session bean and entity bean.	It is a stateless component and accessed by one client at a time.
Portability	It is portable across multiple platforms.	It is a middleware component for windows NT, so it is portable for NT only.
Resource management	The EJB server maintains all components in memory, and other servers can evict every component after a method call. To manage the resources such as threads and method, it implements EJB server.	To manage the resources such as threads and method, it implements MTS run time. It enables the component to operate automatically in multiuser environment without forcing the developer.
Persistence management	It has both persistent and nonpersistent components.	The MTS components are persistent, and it allows the developer to invent and implement their own persistent algorithm.
State management	It uses both stateful and stateless model.	It uses a stateless model to ensure server resource management.

2.5 Indexed Services of XA Interface Used in DTP

This chapter explains about flags, structure definition, and error codes and also declares about the routines by which RM calls a TM. The “xa.h” header file that has a minimum content is used for common language C and ANSI C implementation. It uses certain

naming convention to name its flags, return codes, and functions (Distributed Transaction Processing: The XA Specification 1991). In “xa.h,” all names that appear are part of the namespace, and XA naming conventions are described below:

- In xa_ negative codes, routines have begun with XAER_ and nonnegative return codes are beginning with XA_.
- The TM function that calls RM is written as ax_ (example: ax_reg). The negative codes start with TMER_ and nonnegative return codes as TM_.

1. xa_reg Register an RM with a TM

Synopsis:

```
#include "xa.h"
int ax_reg(int rmid, XID *xid, long flags)
```

Return value: it has the following return values:

- [TM_JOIN]—The resource manager combines the work of the existing transaction branch, and it does not recognize *xid, and then it indicates failure to the application.
 - [TM_OK]—It is normal execution.
 - [TMER_TMERR]—The registration of resource manager an error will occur, which is encountered by the TM by using this.
 - [TMER_INVALID]—Invalid arguments are specified.
 - [TMER_PROTO]—The routine was taken as an improper context.
- #### 2. ax_unreg It will unregister resource manager in a TM.

Synopsis:

```
#include "xa.h"
int ax_unreg(int rmid, long flags)
```

Return value:

- [TM_OK]—Normal execution.
 - [TMER_TMERR]—When a resource manager unregisters the transaction, an error will occur, which is encountered by the TM.
 - [TMER_INVALID]—Invalid arguments are specified.
 - [TMER_PROTO]—The routine was invoked as an improper context.
- #### 3. xa_close close a resource manager.

Synopsis:

```
#include "xa.h"
int xa_close(char *xa_info, int rmid, long flags)
```

Return value:

- [XA_OK]—Normal execution.
- [XAER_ASYNC]—TMASYNC set in flags.
- [XAER_RMERR]—An error occurred when closing resource.

- [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was taken as an improper context.
4. *xa_commit* It commits the work done on behalf of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_commit(XID *xid, int rmid, long flags)
```

Return value:

- [XA_HEURHAZ]—Due to some failures in the specified transaction branch during the work was done, it has been heuristically completed.
 - [XA_HEURCOM]—In this process, a specific transaction branch is committed.
 - [XA_HEURRB]—The specified transaction branch was rolled back.
 - [XA_HEURMIX]—It partially committed specified transaction branch or partially rolled back the transaction.
 - [XA_RETRY]—The resource manager is not be able to commit the transaction at this time. It returns the value when a blocking condition exists and TMNOWAIT was set.
 - [XA_OK]—Normal execution.
 - [XA_RB*]—The resource manager does not commit the transaction.
 - [XA_RBROLLBACK]—For an unspecified reason, the resource manager rolls back the transaction.
 - [XA_RBCOMMFAIL]—The communication is failing within the resource manager.
 - [XA_RBDEADLOCK]—It detects deadlock.
 - [XA_RBINTEGRITY]—The integrity of its resources is detected by the resource manager.
 - [XA_RBOTHER]—The rollback transaction will occur by resource manager which is not present in the list.
 - [XA_RBPROTO]—The protocol error will occur within the resource manager.
 - [XA_RBTIMEOUT]—The transaction will take too much time to finish the work.
 - [XA_RBTRANSIENT]—A transient error is detected by the resource manager.
 - [XAER_RMERR]—When the transaction commits the work to perform, an error will occur and the branch has been rolled back.
 - [XAER_RMFAIL]—An error occurs which makes resource manager unavailable.
 - [XAER_NOTA]—A specific XID is not known by the resource manager.
 - [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was invoked by improper context.
5. *xa_complete* It waits for an asynchronous operation to complete.
- Synopsis:

```
#include "xa.h"
int xa_complete(int *handle, int *retval, int rmid, long flags)
```

Return value:

- [XA_RETRY]—TMNOWAIT is set in flags, and asynchronous operation has not been completed.
 - [XA_OK]—Normal execution.
 - [XAER_PROTO]—The routine was invoked in an improper context.
 - [XAER_INVALID]—Invalid arguments are specified.
6. *xa_end* The end work is performed on behalf of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_end(XID *xid, int rmid, long flags)
```

Return value:

- [XA_NOMIGRATE]—The resource manager is not responsible for preparing transaction context of migration, but TM can resume the association only in the current thread. A resource manager sets TMNOMIGRATE in the flag element of structure *xa_switch_t*, so there is no need to return [XA_NOMIGRATE].
- [XA_OK]—Normal execution.
- [XA_RB*]—The resource manager disconnects the transaction from thread of control and marks rollback operation which was performed on behalf of *xid. The following values are returned regardless of flag setting:
 - [XA_RBROLLBACK]—This rollback function is only for unspecified reason which is marked by the resource manager.
 - [XA_RBCOMMFAIL]—The communication failure will occur within the resource manager.
 - [XA_RBDEADLOCK]—The deadlock is detected in the resource manager.
 - [XA_RBINTEGRITY]—The integrity of resources is detected by the resource manager.
- [XA_RBOTHER]—The rollback transaction will occur by the resource manager which is not present in the list.
- [XA_RBPROTO]—The protocol error will occur within the resource manager.
- [XA_RBTIMEOUT]—The transaction will take too much time to finish the work.
- [XA_RBTRANSIENT]—A transient error is detected by the resource manager.
 - [XAER_RMERR]—When the transaction commits the work to perform, an error will occur and the branch has been rolled back.
 - [XAER_RMFAIL]—An error occurs which makes the resource manager unavailable.
 - [XAER_NOTA]—A specific XID is not known by the resource manager.
 - [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was invoked by improper context.

7. *xa_forget* It forgets about the historic completion of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_forget(XID *xid, int rmid, long flags)
```

Return value:

- [XA_OK]—Normal execution.
- [XAER_RMERR]—An error will occur in the resource manager, and it will not forget about transaction branch.
- [XAER_RMFAIL]—An error occurs which makes the resource manager unavailable.
- [XAER_NOTA]—A specific XID is not known by the resource manager as a historically completed xid.
- [XAER_INVAL]—Invalid arguments are specified.
- [XAER_PROTO]—The routine was invoked by improper context.

8. *xa_open* The *resource* manager is opened.

Synopsis:

```
#include "xa.h"
int xa_open(char *xa_info, int rmid, long flags)
```

Return value:

- [XA_OK]—Normal execution.
- [XAER_RMERR]—An error will occur in the resource manager, and it will not forget about transaction branch.
- [XAER_RMFAIL]—An error occurs which makes the resource manager unavailable.
- [XAER_NOTA]—A specific XID is not known by the resource manager as a historically completed xid.
- [XAER_INVAL]—Invalid arguments are specified.
- [XAER_PROTO]—The routine was invoked by improper context.

9. *xa_prepare* It *prepares* the commit operation to work in front of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_prepare(XID *xid, int rmid, long flags)
```

Return value:

- [XA_RDONLY]—The transaction is committed, and read-only operation is possible.
- [XA_OK]—Normal execution.
- [XA_RB*]—The resource manager does not commit the work to be done on behalf of transaction branch, and it rolls back the branch work to release all resources which was held. The following values may be returned:

- [XA_RBROLLBACK]—This rollback function is only for unspecified reason which is marked by the resource manager.
 - [XA_RBCOMMFAIL]—The communication failure will occur within the resource manager.
 - [XA_RBDEADLOCK]—The deadlock is detected in the resource manager.
 - [XA_RBINTEGRITY]—The integrity of resources is detected by the resource manager.
 - [XA_RBOTHER]—The rollback transaction will occur by the resource manager which is not present in the list.
 - [XA_RBPROTO]—The protocol error will occur within the resource manager.
 - [XA_RBTIMEOUT]—The transaction will take too much time to finish the work.
 - [XA_RBTRANSIENT]—A transient error is detected by the resource manager.
 - [XAER_ASYNC]—TMASYNC is set in flags; either the maximum number of asynchronous operations has been exceeded.
 - [XAER_RMERR]—When the resource manager prepares to commit the work for transaction branch, an error will be encountered.
 - [XAER_RMFAIL]—An error occurs which makes the resource manager unavailable.
 - [XAER_NOTA]—A specific XID is not known by the resource manager.
 - [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was invoked by improper context.
10. *xa_recover* This process will obtain a list of prepared transactions from the resource manager.

Synopsis:

```
#include "xa.h"
int xa_recover(XID *xids, long count, int rmid, long flags)
```

Return value:

- [XAER_RMERR]—An error will be occurred by determining XIDs to return.
 - [XAER_RMFAIL]—An error occurs which makes the resource manager unavailable.
 - [XAER_INVALID]—Invalid flags are specified when pointer XIDs are null and count is negative or greater than zero.
 - [XAER_PROTO]—The routine was invoked by improper context.
11. *xa_rollback* The rollback operation is done on behalf of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_rollback(XID *xids, int rmid, long flags)
```

Return value:

- [XA_HEURHAZ]—Due to some failure in the specified transaction branch during the work was done, it has been heuristically completed. The resource manager will return a value only it successfully prepare *xid.

- [XA_HEURCOM]—In this process, specified transaction branch was committed. The resource manager will return a value only it successfully prepare *xid.
- [XA_HEURRB]—The specified transaction branch was rolled back. The resource manager will return a value only it successfully prepares *xid.
- [XA_HEURMIX]—The specified transaction branch was partially committed and partially rolled back.
 - [XA_OK]—Normal execution.
 - [XA_RB*]—The resource manager does not commit the work to be done on behalf of the transaction branch, and it rolls back the branch work to release all resources which was held. The following values may be returned:
 - [XA_RBROLLBACK]—This rollback function is only for unspecified reason which is marked by the resource manager.
 - [XA_RBCOMMFAIL]—The communication failure will occur within the resource manager.
 - [XA_RVBDEADLOCK]—The deadlock is detected in the resource manager.
 - [XA_RBINTEGRITY]—The integrity of resources is detected by the resource manager.
- [XA_RBOTHER]—The rollback transaction will occur by the resource manager which is not present in the list.
- [XA_RBPROTO]—The protocol error will occur within the resource manager.
- [XA_RBTIMEOUT]—The transaction will take too much time to finish the work.
- [XA_RBTRANSIENT]—A transient error is detected by the resource manager.
 - [XAER_ASYNC]—TMASYNC is set in flags; either the maximum number of asynchronous operations has been exceeded.
 - [XAER_RMERR]—The resource manager will forget about the transaction branch when an error occurs during rollback operation.
 - [XAER_RMFAIL]—An error occurs which makes resource manager unavailable.
 - [XAER_NOTA]—A specific XID is not known by the resource manager.
 - [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was invoked by improper context.

12. *xa_start* It **starts** the work on behalf of a transaction branch.

Synopsis:

```
#include "xa.h"
int xa_start(XID *xid, int rmid, long flags)
```

Return value:

- [XA_OK]—Normal execution.
- [XA_RB*]—It does not associate transaction branch with thread of control, and it marked *xid rollback only. The following values are returned:

- [XA_RBROLLBACK]—This rollback function is only for unspecified reason which is marked by the resource manager.
- [XA_RBCOMMFAIL]—The communication failure will occur within resource manager.
- [XA_RBDEADLOCK]—The deadlock is detected in the resource manager.
- [XA_RBINTEGRITY]—The integrity of resources is detected by the resource manager.
- [XA_RBOTHER]—The rollback transaction will occur by the resource manager which is not present in the list.
- [XA_RBPROTO]—The protocol error will occur within the resource manager.
- [XA_RBTIMEOUT]—The transaction will take too much time to finish the work.
- [XA_RBTRANSIENT]—A transient error is detected by the resource manager.
 - [XAER_ASYNC]—TMASYNC is set in flags; either the maximum number of asynchronous operations has been exceeded.
 - [XAER_RMERR]—The resource manager will forget about the transaction branch when an error occurs during rollback operation.
 - [XAER_RMFAIL]—An error occurs which makes resource manager unavailable.
 - [XAER_DUPID]—The resource manager does not associate the transaction with thread of control. Either TMRESUME or TMJOIN is set in the flags by indicating the initial use of *xid, and XID is already present within the resource manager, and it also returns the value [XAER_DUPID].
 - [XAER_OUTSIDE]—The resource manager will do the work outside of any global transaction.
 - [XAER_NOTA]—Either TMJOIN or TMRESUME is set in flag, and resource manager does not know about specific XID.
 - [XAER_INVALID]—Invalid arguments are specified.
 - [XAER_PROTO]—The routine was invoked by improper context.

2.6 Research Directions: NoSQL DB for Repository Design for IoT

The data storage in Internet of Things (IoTs) is difficult for large database based on the analysis. The test in IoT is to keep up heterogeneous and gigantic information. Kang et al. (2016) by utilizing NoSQL actualized the outline of MongoDB-based EPICS archive. It viably coordinates and stores the heterogeneous IoT information sources, for example, RFID, sensor, and GPS, by developing occasion sorts EPICS. Besides, the information using so as to stockpile in IoT is overseen HBase database in view of the RFID/sensor. The procedure of HBase is powerful in nature, that is, information can be included progressively amid the database exchange.

References

- Bernstein, P. A. and Newcomer, E. 2009. *Principles of Transaction Processing*. San Francisco, CA: Morgan Kaufmann.
- Distributed Transaction Processing: The XA Specification. 1991, December. Retrieved June 26, 2015, from <http://pubs.opengroup.org/onlinepubs/009680699/toc.pdf>.
- Gottlob, G. and Vardi, M. Y. 1995. *Database theory—ICDT '95: 5th International Conference*, Prague, Czech Republic, January 11–13, 1995: *Proceedings*. Berlin: Springer-Verlag.
- Gray, S. D. and Lievano, R. A. 1997. *Microsoft Transaction Server*. Indianapolis, IN: Sams.
- Il-Chul Yoon, Hyung-Ho Kim, Cheong Youn, and Doo-Hwan Bae. 2000. Reliable Transaction Design Using MTS. <http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=884748>.
- Kang, Y., Park, I., Rhee, J. and Lee, Y. 2016. MongoDB-based repository design for IoT-generated RFID/Sensor big data. *IEEE Sensors IEEE Sensors Journal*, 16(2), 485–497. DOI: 10.1109/jsen.2015.2483499.
- Mannai, D. and Elmagarmid, A. 1988. Design and implementation of a distributed transaction processing system. Digest of Papers. *COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference*. DOI: 10.1109/cmpcon.1988.4856.
- Sheldon, T. 2001. *McGraw-Hill Encyclopedia of Networking & Telecommunications*. Berkeley, CA: Osborne.
- Singh, M. P. and Huhns, M. N. 2005. Transaction concepts. In: *Service-Oriented Computing Semantics, Processes, Agents*. Chichester: Wiley, pp. 193–223. DOI: 10.1002/0470091509.ch11.
- Sriganesh, R. P., Brose, G. and Silverman, M. 2006. *Mastering Enterprise JavaBeans 3.0*. Indianapolis, IN: Wiley Pub. <http://media.techtarget.com/tss/static/books/wiley/masteringEJB3/downloads/MasteringEJB4thEd.pdf>.
- Vohra, D. 2013. Using JDBC Data Source with a JNDI. In: *JRuby Rails Web Application Development Springer Briefs in Computer Science*. Heidelberg: Springer, 53–55. DOI: 10.1007/978-3-319-03934-3_13. <https://books.google.co.in/books?isbn=3319039342>.

3

Data Migration Techniques from SQL to NoSQL

Mydhili K. Nair, Nihal Nayak, and Arjun Rao

CONTENTS

3.1	Introduction	48
3.2	Need for Data Migration: A Case Study-Based Analysis	48
3.2.1	RDBMS: Hitting the Glass Ceiling of Data Management Limits	48
3.2.2	Case Study 1: CISCO	49
3.2.3	Case Study 2: Walmart: The World’s Biggest Retailer	50
3.2.4	Case Study 3: Dice.com: IT Job Posting Website.....	50
3.2.5	Case Study 4: eBay: World Renowned Retailer	51
3.3	SQL to MongoDB: Data Migration Using Mongify	52
3.3.1	Overview	52
3.3.2	Installing and Configuring Mongify	52
3.3.3	Importing Data from MySQL Using Mongify.....	52
3.3.4	Importing Specific Rows or Columns from MySQL	55
3.3.5	Incremental Imports	57
3.3.6	Conclusion	58
3.4	Data Migration from MySQL to Hadoop: Using Apache Sqoop	58
3.4.1	Overview	58
3.4.2	Installing and Configuring Apache Sqoop	58
3.4.3	Importing Data from MySQL Using Apache Sqoop	60
3.4.4	Importing Specific Rows or Columns.....	62
3.4.5	Free-Form Query Imports and Incremental Imports	63
3.4.6	Conclusion	63
3.5	SQL to Neo4J: Data Migration Using Neo4J.....	64
3.5.1	Overview	64
3.5.2	Installing and Configuring Neo4J.....	64
3.5.3	Importing Data from MySQL Using Neo4J	64
3.5.4	Importing Specific Rows or Columns.....	68
3.5.5	Use Case for Joining Two Tables.....	69
3.5.6	Incremental Imports	69
3.5.7	Conclusion	69
3.6	Data Migration: Current and Future Trends	69
3.6.1	Overview of Google Cloud Dataflow for Pipelining Data across Storage Systems	69
3.6.2	Other Independent Tools or Solutions for Data Migration.....	70
3.7	Practice Exercises	70
	References.....	72

3.1 Introduction

At the outset, we would like to point out that this chapter assumes that the reader

- Is aware of the basic RDBMS concepts and SQL
- Has an understanding of the characteristic features of NoSQL
- Has at least a beginner-level familiarity with the four types of data stores in NoSQL, namely, document, wide-column, key-value, and graph model
- Is aware of the workings of distributed clustered programming environment

This is because this chapter does not deal with materials such as installation and configuration of any of the NoSQL database case studies. Specifically, this chapter deals with tools and techniques used to migrate data from the world of RDBMS/SQL to the world of NoSQL.

Four case studies are described, corresponding to the four kinds of data storage mechanisms in NoSQL, namely, document-oriented, graph-based, key-value stores, and columnar models. The reasons why RDBMS/SQL could not handle the specific needs of the case study applications are detailed, and the analysis of the reason why a particular kind of NoSQL database was chosen for migration is described. This sets the preamble for hand-holding the reader to the next three sections, each covering tools for migration from a particular type of NoSQL database.

The technology/technical terms used in the book chapter are explained wherever they appear or at the “Key Terminology & Definitions” section. Apart from regular references, additional references are included in the “References for Advance/Further Reading” for the benefit of advanced readers.

3.2 Need for Data Migration: A Case Study-Based Analysis

3.2.1 RDBMS: Hitting the Glass Ceiling of Data Management Limits (Figure 3.1)

“Big Data,” the term used to depict the amount of digital data generated by the electronic devices in today’s world, is almost unfathomable, 2.5 EBs (2.5×10^{18}) per day from 2012 (Rocha et al., 2016; Big Data, n.d.)! This means that the management of data using Extract, Transform, and Load (ETL) operations, involving storing, effectively retrieving, and processing data, is indeed a mammoth task. The problem is augmented by the fact that apart from the volume of data, its type also varies (Keep, 2014) as shown in Table 3.1.

The methodology of application development for the connected web world involving mobile phones, cameras, wireless sensor devices, etc. is usually Agile Development (Keep, 2014), which is innately iterative with short development life cycles. Each of these life cycles needs to store, retrieve, and process the type and volume of data described in Table 3.1. The de facto deployment platform for these applications is the public or private Cloud.

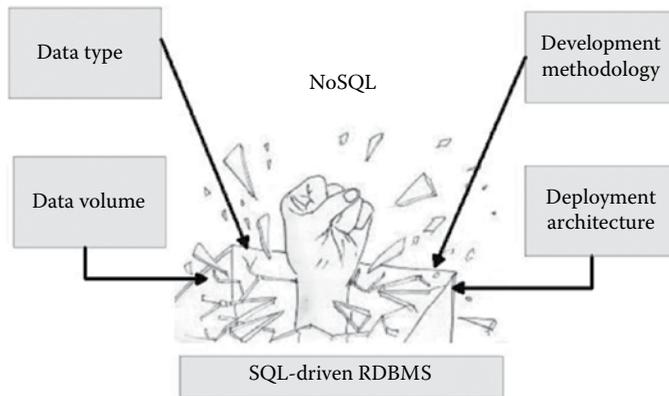


FIGURE 3.1
RDBMS—hitting the glass ceiling of data management limits.

TABLE 3.1
Characteristics of “Big Data”

Type of Data	Volume of Data
<ul style="list-style-type: none"> • Unstructured • Semistructured • Polymorphic 	<ul style="list-style-type: none"> • Petabytes • Millions of records • Trillions of queries per second

Cloud computing architectures demand horizontal scaling, also called “scaling out.” This is because the applications are inherently distributed, many a time, across disparate geographical locations. In horizontal scaling, when there is an increase in application performance expectations and load, additional servers are added. This leads to running multiple instances of the application in parallel on each of these additional servers (Beaumont, 2014). Vertical scaling on the other hand refers to upgrading the existing server with powerful hardware. This is often called “scaling up” and often involves downtime. RDBMS is heavily dependent on SQL for its ETL operations. SQL has huge performance issues when dealing with distributed applications that are “scaled out” as it involves intelligent partitioning of the database, writing optimized set of SQL Join operations, Sub-queries, and Stored Procedures across multiple parallel database instances of the application deployed on a set of servers.

Sections 3.2.2 through 3.2.7 highlight case studies of successful migration from SQL to NoSQL databases. The case studies are chosen so that the SQL to NoSQL migration of all four forms of NoSQL data storage mechanisms, namely, key–value stores, document stores, column stores, and graph store models is dealt with to highlight the need for data migration from the structured to the unstructured domains!

3.2.2 Case Study 1: CISCO

Application: Enterprise Social Networking (Keep, 2014)

Problems faced using SQL-driven RDBMS:

- Complex SQL queries
- Highly normalized schema not aligned to multiple data types
- Very poor horizontal scalability

Migrated to: MongoDB, a document-driven NoSQL database

Reason:

- Dynamic schema creation using JSON (Trivedi, 2014)
- Social Network Analysis using light-weight map-reduce algorithms

Benefits: CISCO could add new enterprise social networking features easily due to flexible, dynamic schema features inherent in JSON. The speed of retrieval of information, read operation, reduced from an average of 30 s to less than 10 ms.

3.2.3 Case Study 2: Walmart: The World's Biggest Retailer

Application: Optimization of Walmart's online recommendations to its customers (Neo4j at Walmart, 2016).

Problems faced using SQL-driven RDBMS:

- Too much delay in producing real-time data aggregation for data analytics
- Highly complex SQL queries
- Unpredictable peak loads

Migrated to: Neo4J, a graph-oriented NoSQL database

Reason:

- Graph databases can quickly query customers' past purchases as well as instantly capture new interests shown in his/her current online visit.
- Matching of historical and current session data facilitates real-time relevant recommendations for the customer.

Benefits: Walmart is now using Neo4j to make sense of online shoppers' behavior to be able to optimize and cross-sell major product lines in core markets. It has deployed Neo4j in its remarketing application, run by the company's eCommerce IT team based in Brazil (Neo4j at Walmart, 2016).

3.2.4 Case Study 3: Dice.com: IT Job Posting Website

Application: To build a highly customized content management system that would allow tens of thousands of articles to be stored and published on the web (Cogswell, 2013).

Problems faced using SQL-driven RDBMS:

- Articles have large metadata attached to them, for example, abstracts, information about publisher, and biographies.
- This metadata is highly unstructured.
- Large quantities of image and video data to be stored, retrieved, and updated without read and write latency.

Migrated to: Amazon Dynamo DB, a key–value pair-oriented database which is hosted NoSQL distributed among Amazon servers.

Reason:

- Eliminates tons of attributes. Instead, use single document per user/article.
- Simple scalability and high availability.

Benefits: Amazon claims that they keep Dynamo DB in solid-state devices which keep speeds fast. Initial deployment held 5 billion documents and 10 TBs of data. Automated failover provides high availability.

3.2.5 Case Study 4: eBay: World Renowned Retailer

Application: To build a system that understands each person’s unique habits and preferences and brings to light products and items that a user may be unaware of and not looking for (Datastax—eBay, 2014).

Problems faced using SQL-driven RDBMS:

- Not scalable to handle billions of reads and writes each day to be processed at blistering speeds.
- ACID transactional constraints of traditional RDBMS.
- Not efficient to do rapid analysis on a broad assortment of structured and unstructured data captured real time, while customer is online.

Migrated to: Datastax, an enterprise, commercial version of Apache Cassandra, a hybrid between key–value and column-oriented NoSQL databases.

Reason:

- Robust support for clusters spanning multiple data centers
- Encourages data redundancy through data replication
- High availability and no scope of single-point failure.

Benefits: eBay is able to store vast amounts of data around 250 TBs and handle extreme data velocity with six billion writes and five billion reads daily. The company is able to deliver highly accurate and personalized search results with low latency, ensuring 100% uptime, even during peak data loads.

3.3 SQL to MongoDB: Data Migration Using Mongify

3.3.1 Overview

Mongify is a database translator for SQL to MongoDB. Mongify is completely an open source. Mongify was created by Andrew Kalek. It supports MySQL and SQLite. But the creator claims that it supports other SQL databases as well (Anlek, 2016)! In this chapter, migration from MySQL to MongoDB is detailed.

3.3.2 Installing and Configuring Mongify

Since the case study scenario is migrating from SQL to MongoDB, it is assumed that the reader has MySQL installed with the set of tables to transform to NoSQL MongoDB ready! MongoDB and Mongify can be downloaded from the following links:

Download URL for MongoDB: <https://www.mongodb.org/downloads>

Download URL for Mongify: <https://github.com/anlek/mongify>.

Official website: <http://mongify.com>

Mongify is RubyGems software. The RubyGems software allows you to easily download, install, and use ruby software packages on your system. The software package is called a “gem,” and it contains a package Ruby application or library. Gems can be used to extend or modify functionality in Ruby applications. Commonly, they are used to distribute reusable functionality that is shared with other Rubyists for use in their applications and libraries. Some gems provide command-line utilities to help automate tasks and speed up your work. It is available on the “gem” package manager. To install Mongify execute, the following command is used:

```
$ gem install mongify
```

3.3.3 Importing Data from MySQL Using Mongify

Use Case: Two tables in “Foreign Key” referencing relationship

Table 3.2 depicts two MySQL tables “Person” and “FavoriteFood” which have to be migrated to MongoDB. The “Person” table has three attributes: “pid,” “firstname,” and “lastname.” The primary key in “Person” is “pid.” “FavoriteFood” is another table which has three attributes: “fid,” “food,” and “pid.” The primary key in “FavoriteFood” is “fid,” and “pid” is the foreign key referenced from “Person” table. The two tables are part of the “FoodLove” database.

Case 1: Each person likes one food item

The process of converting SQL to MongoDB is broken down into the following steps:

- *Step 1:* Connect Mongify to SQL and NoSQL databases. Here, they are MySQL and MongoDB, respectively.
- *Step 2:* Generate the translation file. Translation file is a ruby file that provides mapping instructions to Mongify.
- *Step 3:* Execute commands to convert the SQL rows to MongoDB documents.

TABLE 3.2

Case 1—Data in “Person” and “FavoriteFood” MySQL Tables

<i>Person</i>		
pid	firstname	lastname
1	Lionel	White
2	Mary	Lambert
3	Alia	Lee
<i>FavoriteFood</i>		
fid	food	pid
1	Bacon	1
2	Fries	2
3	Fish & Chips	3

Follow the steps to migrate using Mongify:

- *Step 1: Establishing connection—execution and testing:* Mongify needs to be connected with SQL and NoSQL databases. For this, let us create a configuration file—`database.config`. A snapshot of the contents of this configuration file is given in [Table 3.3](#).

To check if connection has been established, execute the following command:

```
$ mongify check database.config
```

You should be able to see the following messages on your terminal.

```
SQL connection works
NoSQL connection works
```

TABLE 3.3Entries in `database.config`

```
sql_connection do
  adapter "mysql"
  host    "localhost"
  username "root"
  password "password"
  database "FoodLove"
  batch_size 10000
end
mongodb_connection do
  host    "localhost"
  database "Food"
end
```

If you get an error, then check the entries in your `database.config` file and also if it has the correct access and execute permissions.

- *Step 2: Generating translation file—execution and testing:* Next step is to create the translation file. To automatically create the translation file, execute the following command:

```
$ mongify translation database.config > translation_food.rb
```

This pipes the translation into the file `translation_food.rb`. This creates a basic structure without any sort of referencing or embedding, as shown in [Figure 3.2](#).

We need to edit `translation_food.rb` to suit the way we want to store the documents. In our example, “FavoriteFood” is embedded in “Person” ([Table 3.4](#)).

```
translation_food.rb x database.config x
table "FavoriteFood", :embed_in => :Person, :on => :pid do
  column "fid", :key, :references => :Person
  column "food", :string
  column "pid", :integer
end

table "Person" do
  column "pid", :key
  column "firstname", :string
  column "lastname", :string
end
```

FIGURE 3.2
Entries in `translation_food.rb`.

TABLE 3.4

Commands to Edit Translation File as per Use Case

```
table "FavoriteFood", :embed_in => :Person, :on => :pid do
  column "fid", :key, :references => :Person
  column "food", :string
  column "pid", :integer
end

table "Person" do
  column "pid", :key
  column "firstname", :string
  column "lastname", :string
end
```

TABLE 3.5

Snapshot of MongoDB Document after Migration

```

{
  "_id" : ObjectId("56cdc0874e05e51a9c000001"),
  "firstname" : "Lionel",
  "lastname" : "White",
  "FavoriteFood" : [
    {
      "food" : "Bacon"
    }
  ]
}

```

Mongify provides several options to help the translation. In our example, we have used “embed_in” and “on,” which are options for tables, and references is an option for columns. The “embed_in” option embeds the table in another table. This feature of embedding serves as an alternative for the SQL Join. In our example, we are embedding “FavoriteFood” in “Person” and linking the two using the option “on.” The parameter of “on” option indicates Mongify linking between two tables.

- *Step 3: Conversion—execution and testing:* Lastly, we need to type the command that magically converts the entire MySQL database to MongoDB.

```
$ mongify process database.config translation_food.rb
```

The entire process happens quickly, and the end result looks something as shown in (Table 3.5).

The result will be an array embedded within the resulting document, as shown in Figure 3.3.

Case 2: A person can like multiple food items. In Table 3.6, *Mary* likes *Fries* and *Nachos*.

The basic procedure for translating SQL to MongoDB remains the same as described in Case 1, even after we added a few rows in MySQL and made *Mary* like *Nachos* apart from *Fries*. In Figure 3.4, you will see that two attributes of food are embedded in *Mary’s* document.

Was not that simple? Now you can go ahead and try to implement the same procedure for other MySQL tables.

3.3.4 Importing Specific Rows or Columns from MySQL

This tool is not perfect. The reason is that the conditional importing from SQL to MongoDB is not straightforward. One needs to create another table or a view and include that view

```

> db.Person.find()
{ "_id" : ObjectId("56cdc0874e05e51a9c000001"), "firstname" : "Lionel", "lastname" : "White", "FavoriteFood" : [ { "food" : "Bacon" } ] }
{ "_id" : ObjectId("56cdc0874e05e51a9c000002"), "firstname" : "Mary", "lastname" : "Lambert", "FavoriteFood" : [ { "food" : "Fries" } ] }
{ "_id" : ObjectId("56cdc0874e05e51a9c000003"), "firstname" : "Alia", "lastname" : "Lee", "FavoriteFood" : [ { "food" : "Fish & Chips" } ] }
>

```

FIGURE 3.3

Snapshot of the result document.

TABLE 3.6

Case 2—Modified Data in Person and FavoriteFood MySQL Tables

<i>Person</i>		
pid	firstname	lastname
1	Lionel	White
2	Mary	Lambert
3	Alia	Lee
4	Tom	Joseph

<i>FavoriteFood</i>		
fid	food	pid
1	Bacon	1
2	Fries	2
3	Fish & Chips	3
4	Tacos	4
5	Nachos	2

in the translation file. The reason is that Mongify was created in order to migrate the entire SQL database to MongoDB. There is a method in MongoDB called “before_save” used to make amends to the columns or rows before saving, but this involves manually updating the contents of the database to reflect the changes to its data in the original MySQL version. There is no direct command or method to only import specific rows and columns. The entire MySQL database gets imported. *This is one of the unresolved issues in the tool.*

The present method of importing specific rows and columns is detailed in [Tables 3.7](#) through [3.9](#).

When we execute the command mongify process, the result obtained is shown in [Figure 3.5](#).

You might have noticed that “FavoriteFood” attribute is missing from the new document. It has been replaced by “Mary” attribute because the name of the table/view is

```
> db.Person.find()
{ "_id" : ObjectId("56fbc8b84e05e51ab0000001"), "firstname" : "Lionel", "lastname" : "White", "FavoriteFood" : [ { "food" : "Bacon" } ] }
{ "_id" : ObjectId("56fbc8b84e05e51ab0000002"), "firstname" : "Mary", "lastname" : "Lambert", "FavoriteFood" : [ { "food" : "Fries" }, { "food" : "Nachos" } ] }
{ "_id" : ObjectId("56fbc8b84e05e51ab0000003"), "firstname" : "Alia", "lastname" : "Lee", "FavoriteFood" : [ { "food" : "Fish & Chips" } ] }
{ "_id" : ObjectId("56fbc8b84e05e51ab0000004"), "firstname" : "Tom", "lastname" : "Joseph", "FavoriteFood" : [ { "food" : "Tacos" } ] }
```

FIGURE 3.4

Embedded MongoDB document for Case 2—Mary likes Fries and Nachos.

TABLE 3.7

Creating a View for Mary from Person Table

```
> create view MaryPerson as select * from Person where pid =2;
```

TABLE 3.8

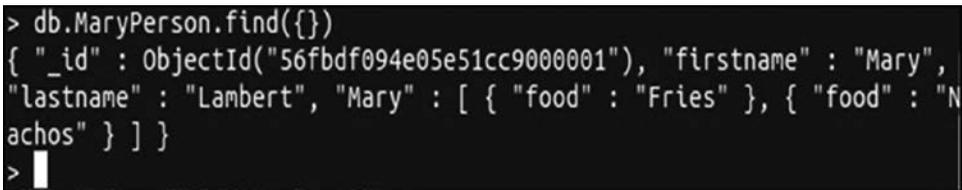
Creating a View for Mary from Favorite Table

```
> create view Mary as select * from FavoriteFood where pid =2;
```

TABLE 3.9

Modify the Translation File to Suit Mary's Views

```
table "Mary", :embed_in => :MaryPerson, :on => :pid do
  column "fid", :key, :references => :MaryPerson
  column "food", :string
  column "pid", :integer
end
table "MaryPerson" do
  column "pid", :key
  column "firstname", :string
  column "lastname", :string
end
```



```
> db.MaryPerson.find({})
{ "_id" : ObjectId("56fbdf094e05e51cc9000001"), "firstname" : "Mary",
"lastname" : "Lambert", "Mary" : [ { "food" : "Fries" }, { "food" : "N
achos" } ] }
> |
```

FIGURE 3.5

Creating a view for Mary from Favorite Table.

"Mary." You can use the "rename_to" property in your translation file to change it to "FavoriteFood" or anything else that may suit your application (Table 3.10).

3.3.5 Incremental Imports

Mongify's "sync" command is used for incremental imports. Usage of this command is depicted in Table 3.11. If you want to continually sync your SQL database with MongoDB using Mongify, you MUST run the "sync" feature from the start, that is, you must execute the sync command before executing the process command. Using the process command will not permit future syncing. The "sync" feature leaves the "pre_mongify_id" in

TABLE 3.10

Syntax for Renaming a Table in Translation File

```
table "table_name", :rename_to => 'my_table'
```

TABLE 3.11

Usage of Sync Feature

```
mongify sync database.config database_translation.rb
```

your documents, as well as it creates its own collection called *“mongify_sync_helper”* to keep dates and times of your last sync.

3.3.6 Conclusion

Mongify tool is an open-source tool, and therefore it is easily accessible to the community. Mongify helped us import an SQL database to MongoDB database. This is a powerful tool indeed. However, we have also seen the flaws in the tool with respect to importing specific rows using WHERE condition. Since the source code is available on Mongify’s Github repository (Anlek, 2016), developers can enhance the present version. We have also mentioned several proprietary tools in the references section later in this chapter (Section 3.6) which help in faster migration.

3.4 Data Migration from MySQL to Hadoop: Using Apache Sqoop

3.4.1 Overview

The motivation for data migration from MySQL to a Hadoop cluster is two-fold. The data present in the MySQL database are too large to be processed through an SQL query in a reasonable amount of time, or the data are infrequently accessed and it is better to store these large data in Hadoop, which is optimized for dealing with storing large datasets like usage logs, error logs, etc. In this section, we will be considering data migration from a MySQL database to a Hadoop cluster; for purposes of this chapter, we will look at a simple Hadoop configuration running in a pseudo-distributed mode on a single computer. However, the process is the same for a fully distributed Hadoop cluster.

Data migration from MySQL to Hadoop can be done manually by exporting SQL data into a format like CSV, and importing the generated CSV into HDFS using any simple scripting language like Perl, Python, or Ruby. While such a method is possible given that most SQL databases include an option to export data as CSV, it becomes difficult when we would like to parse multiple tables at one go or would like to automate the process for a large database or a set of databases that need migration.

Another approach is to write a custom tool for migrating data into Hadoop that is tailored to the requirements of the given situation. This can be particularly useful if data need to be processed while maintaining some structure to the data that is drawn as a result of performing SQL joins on multiple tables. If we were to store the result of multiple joins as a set of JSON objects, we can load these JSON objects stored in JSON files directly into Hadoop’s file system by using the Hadoop’s in-built command-line utilities (Apache Hadoop 2.7.2, 2016).

In this section, however, we will be discussing the use of Apache Sqoop as a tool for data migration from MySQL to Hadoop. While the reverse is possible, that is, to use Sqoop to export data from Hadoop to MySQL, we will not be discussing that here and it is left as an exercise for the reader as it is fairly straightforward.

3.4.2 Installing and Configuring Apache Sqoop

Sqoop requires Java to be installed prior to its installation. Once the latest version of JDK (JavaSE Downloads, 2016) is installed and is available in the system path, it is required to set up environment variables for Sqoop’s configuration. An environment variable pointing

to the java installation is also required, which must be named as "JAVA_HOME" pointing to the install path of the JDK.

The command for this on Linux/Unix operating systems is as follows:

```
$ export JAVA_HOME=<path to java installation>
```

Apart from Java, Hadoop must be installed on the system, and running the following command can check this:

```
$ hadoop version
```

Once Hadoop is configured correctly, and running, we will be able to use Hadoop's web interface to view all applications on the cluster.

Apache Sqoop is available for download through various mirrors that can be found at this link: <http://sqoop.apache.org>

The compiled binary version of Apache Sqoop can be downloaded as a tar .gz file and extracted into any suitable directory.

To configure Sqoop, we will need to set up an environment variable called "SQOOP_HOME" pointing to the directory where the downloaded file was extracted. This can be done using the following command:

```
$ export SQOOP_HOME=<path to sqoop directory>
```

We are also required to edit the file named sqoop-env.sh in the \$SQOOP_HOME/conf directory. There is a template file for sqoop-env named as sqoop-env-template.sh that can be renamed to sqoop-env.sh and edited. The following lines in the file need to be changed:

```
$export HADOOP_COMMON_HOME=<path to hadoop directory>
export HADOOP_MAPRED_HOME=<path to hadoop directory>
```

These need to be set up as per the Hadoop cluster configuration on the system.

To connect MySQL with Sqoop, we will require the mysql-j-connector library, which is the official JDBC driver for MySQL. It can be downloaded from: <https://dev.mysql.com/downloads/connector/j/>

The downloaded file must be extracted, and the jar file containing the j-connector binaries must be placed in \$SQOOP_HOME/lib/ directory.

If all the configuration steps are performed correctly, when running \$ Sqoop version at the command line while in the \$SQOOP_HOME/bin directory, the output that can be expected is similar to:

```
INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
Sqoop 1.4.6
git commit id c0c5a81723759fa575844a0a1eae8f510fa32c25
Compiled by root on Mon Apr 27 14:38:36 CST 2015
```

3.4.3 Importing Data from MySQL Using Apache Sqoop

For the purpose of demonstration, we will be using a sample MySQL database having a table with customer details for a food mart (MySQL Foodmart DB, 2008). The structure of the table is given in [Table 3.12](#). It contains about 10,000 entries of customer information; however, it should be noted that Sqoop is capable of handling datasets of much larger volume. Importing from MySQL to Hadoop using Sqoop requires passing the MySQL database connection parameters to Sqoop along with optional parameters specifying additional constraints or filters on the data being imported.

The general syntax for importing data into HDFS using Sqoop is as follows:

```
$ sqoop import <generic-args><import-args>
```

Import arguments include parameters like the following:

```
--connect <url> (uses the jdbc connection url to connect to MySQL)
--username <username> (Username to access MySQL database)
--P (to prompt for MySQL database password)
--m, --num-mappers <n> (to use n map tasks to import in parallel)
--target-dir <directory> (to change the hdfs directory for storing
results of import)
```

TABLE 3.12
Structure of “Customers” Table in MySQL

Field	Type
customer_id	int(11)
account_num	bigint(20)
lname	varchar(30)
fname	varchar(30)
mname	varchar(30)
address	varchar(30)
city	varchar(30)
state_province	varchar(30)
postal_code	varchar(30)
country	varchar(30)
customer_region_id	int(11)
phone	varchar(30)
birthdate	date
gender	varchar(30)
date_accnt_opened	date
member_card	varchar(30)
fullname	varchar(60)

To import all data from a given table, we can run Sqoop with the following parameters:

```
$ sqoop import \  
--connect jdbc:mysql://localhost/sqoopdb \  
--username root \  
--P\  
--table customers \  
--m 2 \  
--target-dir /resultset
```

The above command will import all records with all attributes from the “customers” table using two parallel map processes and store the result in the /resultset directory in HDFS. If we can check the result by running the following command,

```
$ hdfs dfs -ls /resultset  
Found 3 items  
-rw-r--r-- 1 user supergroup 0 2016-03-29 21:10 /resultset/_SUCCESS  
-rw-r--r-- 1 user supergroup 1423203 2016-03-29 21:10 /resultset/  
part-m-00000  
-rw-r--r-- 1 user supergroup 1423203 2016-03-29 21:10 /resultset/  
part-m-00001
```

This will list all the contents of the directory, and it should contain two files named as part-m-000#, which are basically the results of the two map-reduce processes. They can be viewed by exporting them to a CSV file using a command such as the following:

```
$ hdfs dfs -cat /resultset/part-m-* >~/customerdata.csv
```

As an example, the data obtained are as follows:

```
1,87462024688,Nowmer, Sheri, A, 2433 Bailey Road, Tlaxiaco, Oaxaca,  
15057, Mexico,...  
2,87470586299,Whelply, Derrick, I, 2219 Dewing Avenue, Sooke, BC,  
17172, Canada,...  
3,87475757600,Derry, Jeanne, K, 7640 First Ave., Issaquah, WA,  
73980, USA,...  
...  
...  
...  
10,87500482201,Spence, Michael, J, 337 Tosca Way, Burnaby, BC,  
74674, Canada,...
```

Additional import parameters can be used to change the nature of data being imported and hence configured to import either specific rows or columns or only all tables at one go, etc.

3.4.4 Importing Specific Rows or Columns

To import specific rows, the clause can be used. For example, to import records of customers residing in the United States, we can use the following command:

```
$ sqoop import \  
--connect jdbc:mysql://localhost/sqoopdb \  
--username root \  
--P\  
--table customers \  
--m 2 \  
--where "country = 'USA'" \  
--target-dir /resultset
```

The result set obtained in HDFS (as shown below) will now contain data of customers where the country column is the United States for all records.

```
3,87475757600,Derry,Jeanne,null,7640 First Ave., Issaquah, WA,  
73980, USA,.....  
5,87514054179,Gutierrez,Maya,null,8668 Via Neruda, Novato, CA,  
57355, USA,.....  
6,87517782449,Damstra,Robert,F.,1619 Stillman Court, Lynnwood, WA,  
90792, USA, .....  
...
```

To restrict the import to contain only "account_number, fullname, country" for each record, we can make use of the `--columns` argument. This takes a comma-separated list of columns to be imported. The above example of `account_number`, `fullname`, and `country` can be imported using the following Sqoop command:

```
$ sqoop import \  
--connect jdbc:mysql://localhost/sqoopdb \  
--username root \  
--P\  
--table customers \  
--m 2 \  
--columns "account_num,fullname,country" \  
--target-dir /resultset
```

The result set obtained will only contain the specified three columns after importing the data into Hadoop, as shown below:

```
87462024688,Sheri Nowmer,Mexico  
87470586299,Derrick Whelply,Canada  
87475757600,Jeanne Derry,USA  
...
```

This is particularly useful if processing on Hadoop is required only on a subset of the data and not the entire table.

3.4.5 Free-Form Query Imports and Incremental Imports

Sqoop also supports importing results of arbitrary SQL queries that could include operations such as JOIN or UNION. It is also possible to run Sqoop import on views created from tables. Free-form query imports use the `--query` parameter which takes an SQL query string along with it to specify the nature of the result set being imported. For example:

```
$ sqoop import \  
--connect jdbc:mysql://localhost/sqoopdb \  
--username root \  
--P\  
--query 'SELECT t1.*, t2.* FROM t1 JOIN t2 on (t1.id == t2.id) WHERE $CONDITIONS' \  
--split-by t1.id \  
--target-dir /joinedresultset
```

The above Sqoop command will perform a join on two tables T1 and T2, and the result of this will be imported into HDFS. The “`split-by`” option is used to identify the column to be used when splitting the data during import into different part files.

Incremental Imports is a feature of Sqoop that allows records to be imported into HDFS, which are newer than some number of previously imported records. This requires three arguments to be specified during import, which will specify the column to be checked for the last value of the record after which data import operations must be performed.

The parameters for incremental imports are as follows:

```
--check-column <col> (specifies the column to examine before importing)  
--incremental <mode> (specifies the mood for determining new records)  
--last-value <value> (specifies the value of the check-column beyond  
which import is to be performed)
```

Permissible incremental modes include “`append`” and “`lastmodified.`” When importing data using the “`append`” mode, new rows are imported based on whether the check-column value for new rows is greater than that specified by last-value argument.

When importing data using the “`lastmodified`” mode, new rows are imported based on timestamps stored in the column specified by check-column and the timestamp specified through the last-value argument. Rows whose timestamps in check-column are newer than that mentioned by last-value get imported.

Once incremental import operation is completed, the value to be specified as last-value for the next incremental import gets printed on the screen, and it should be used to ensure that only newer data are imported into HDFS.

3.4.6 Conclusion

In this section, we have only covered a few basic operations for importing data into HDFS from MySQL using Sqoop. It must be noted that Sqoop provides a lot more options for importing data and configuring the nature of imported data in HDFS. Data imported by

Sqoop are automatically mapped to Java or Hive data types (Sqoop Userguide, 2016), but this can be overridden and custom-type mappings can be specified during import using arguments such as `--map-column-java` or `--map-column-hive`.

In this section, we have illustrated importing data into HDFS. Sqoop can also be configured to import data into Hive (Apache Hive™, 2014) for faster querying and processing using HiveQL or into HBase. The `import-all-tables` option in Sqoop allows a set of tables from MySQL to be imported into HDFS, each being stored in a different directory. This is particularly useful if there are multiple tables with single-column primary keys, and we need to import all columns from all tables.

Exporting data from HDFS to MySQL can be performed in a similar way using the Sqoop export option, which takes connection parameters similar to that of Sqoop import.

While it is always more accurate to use a custom-written tool to perform data migration, Sqoop helps simplify the process in most cases where the data migration is not very complex and is straightforward in nature. In subsequent sections of this chapter, we will see data migration into other NoSQL databases from RDBMS and understand the process of migrating to various NoSQL-based tools.

3.5 SQL to Neo4J: Data Migration Using Neo4J

3.5.1 Overview

Neo4J is an open-source NoSQL database which is built on Java and Scala. The query language used for Neo4J is called “Cypher.” The development of Neo4J started in 2003, and it was publicly made available since 2007. The source code and issue tracking are available on “Github,” with support readily available on “Stack Overflow” and the “Neo4J Google Group.”

Neo4J implements the “Property Graph Model” even at the storage level. As opposed to graph processing or in-memory libraries, Neo4J provides full database characteristics including ACID transaction compliance, cluster support, and runtime failover, making it suitable to use graph data in production scenarios.

The property graph contains connected entities (the nodes) which can hold any number of attributes (key–value pairs). Nodes can be tagged with labels representing their different roles in your domain (Sqoop Userguide, 2016). In addition to contextualizing node and relationship properties, labels may also serve to attach metadata—index or constraint information—to certain nodes (Figure 3.6).

3.5.2 Installing and Configuring Neo4J

Download URL for Neo4J: <http://neo4j.com/download/>

Installing and configuring Neo4J do not come under the scope of this chapter as here we presume that the reader has MySQL and Neo4J installed and wants to know the intricacies of how to migrate their data from MySQL to Neo4J.

3.5.3 Importing Data from MySQL Using Neo4J

Use Case: Two tables in “Foreign Key” referencing relationship

In this section, we will consider two tables, “Desserts” and “Person.” The table “Person” contains “d_id” as a foreign key, referencing to the “d_id” in “Desserts” (Table 3.13).

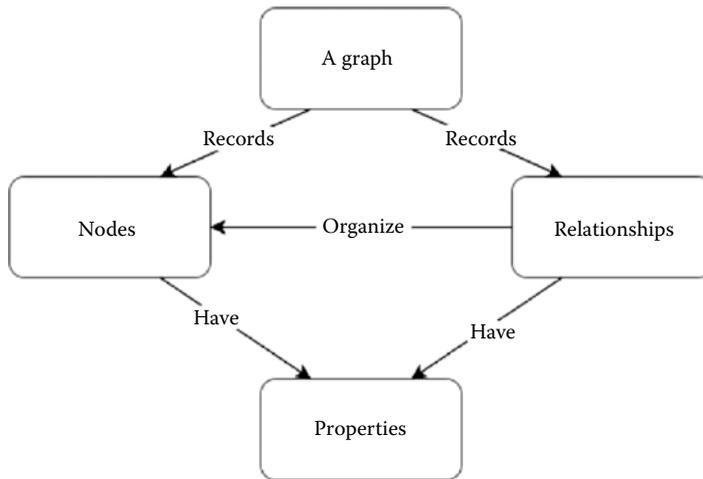


FIGURE 3.6 Property Graph Model. (Adapted from What is Graph Database?, n.d. <https://neo4j.com/developer/graph-database/#property>)

TABLE 3.13

Data in “Person” and “Desserts” MySQL Tables

<i>Desserts</i>			
d_id	itemname		
1	Apple Pie		
2	Chocolate Mousse		
<i>Person</i>			
p_id	firstname	lastname	d_id
1	John	Stone	1
2	Misty	Rose	2
3	LaShawn	Merritt	1
4	Elijah	Wood	2

The migration from SQL to Neo4j is pretty straightforward as Cypher provides a LOAD CSV clause which can be used to import millions of entries to the Neo4j database. We know that SQL tables can be exported as CSV. Therefore, we can break down the procedure into three steps:

- *Step 1:* Exporting SQL table to CSV with headers
- *Step 2:* Import/load CSV to Neo4j
- *Step 3:* Add relationships to the nodes

Follow the steps to migrate:

- *Step 1: Exporting SQL table to CSV with headers—execution and testing:* First step is to export the MySQL tables to CSV. Note that, it can be from any other SQL database as well. For the Use Case taken, the query in [Table 3.14](#) will export the tables with the headers into CSV files.

The query should be entered in the MySQL terminal. The query mentioned in [Table 3.14](#) is projecting the tables into their respective CSV files with comma as the field delimiter. Each line in the CSV file is terminated by carriage return and newline escape characters, as shown in [Tables 3.15](#) and [3.16](#).

- *Step 2: Import/load CSV to Neo4j—execution and testing:* Second step would be load these two CSV files and create nodes for these in Neo4j. We will be iterating through each line of the CSV file as each line will be represented as a node in Neo4j. If you have already obtained a CSV file from an external source, remember to always validate it using a CSV validation tool. We would be creating two labels: “Person” and “Desserts”. Execute the code in the Neo4j terminal to load the CSVs to their respective labels ([Tables 3.17](#) and [3.18](#)).
- *Step 3: Add relationships to the nodes—execution and testing:* Now that we have the nodes, all we need to do is add the relationship using MERGE or CREATE. MERGE is a far better choice as duplicate relationships will not be created if they already exist. We use MERGE in this Use Case ([Table 3.19](#)).

TABLE 3.14

Query to Project the Tables into CSV Files

```

SELECT 'D_ID', 'ITEMNAME'
UNION
SELECT * FROM Desserts
INTO OUTFILE 'desserts.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';
SELECT 'P_ID', 'FIRSTNAME', 'LASTNAME', 'D_ID'
UNION
SELECT * FROM Person
INTO OUTFILE 'person.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

```

TABLE 3.15

person.csv

```

"P_ID", "FIRSTNAME", "LASTNAME", "D_ID"
"1", "John", "Stone", "1"
"2", "Misty", "Rose", "2"
"3", "LaShawn", "Merrit", "1"
"4", "Elijah", "Wood", "2"

```

TABLE 3.16

desserts.csv

```

"D_ID", "ITEMNAME"
"1", "Apple Pie"
"2", "Chocolate Mousse"

```

TABLE 3.17Importing `person.csv` into Neo4j as Person

```
LOAD CSV WITH HEADERS FROM "file:///tmp/person.csv" AS line
CREATE (:Person{pid:TOINT(line.P_ID),
firstname:line.FIRSTNAME,lastname:line.LASTNAME,did:TOINT(line.D_ID)});
```

TABLE 3.18Importing `desserts.csv` into Neo4j as Desserts

```
LOAD CSV WITH HEADERS FROM "file:///tmp/dessert.csv" AS line
CREATE (:Desserts{did:TOINT(line.D_ID), itemname:line.ITEMNAME });
```

TABLE 3.19

Adding Relationship “EATS” Using MERGE

```
LOAD CSV WITH HEADERS FROM "file:///tmp/person.csv" AS csvLine
MATCH (desserts:Desserts { did: TOINT(csvLine.D_ID)})
MATCH (person:Person { did: TOINT(csvLine.D_ID)})
MERGE (person)-[:EATS]->(dessert)
return person,dessert;
```

Notice that we are again loading “`person.csv`” into Neo4j. This is because “`person.csv`” contains the header “`D_ID`” which is the foreign key entity in the “`Person`” table which we have exported to the “`person.csv`” file. The query will help a lot in the scenario when there are many to many relationship CSV schemas.

We are adding the relationship “EATS” to the nodes. The query will yield the graph as shown in [Figure 3.7](#). The figure shows that two nodes “EATS” chocolate mousse relationship and two nodes with “EATS” Apple Pie relationship ([Figure 3.7](#)).

That is “Misty EATS Chocolate Mousse”

“Elijah EATS Chocolate Mousse”

“John EATS Apple Pie”

“LeShawn EATS Apple Pie”

Although this example has only two tables, this can be scaled up to multiple tables with higher complexities. When you are dealing with large sets of data, it is advisable to use “`USING PERIODIC COMMIT.`” Cypher query may fail and manifest itself as `OutOfMemoryError` (Neo4j Developer Manual v3.0, 2016). Cypher provides the global “`USING PERIODIC COMMIT`” query hint for updating queries using `LOAD CSV`. You can optionally set the limit for the number of rows per commit like so: `USING PERIODIC COMMIT 500.`

`PERIODIC COMMIT` will process the rows until the number of rows reaches a limit. Then, the current transaction will be committed and replaced with a newly opened transaction. If no limit is set, a default value will be used.

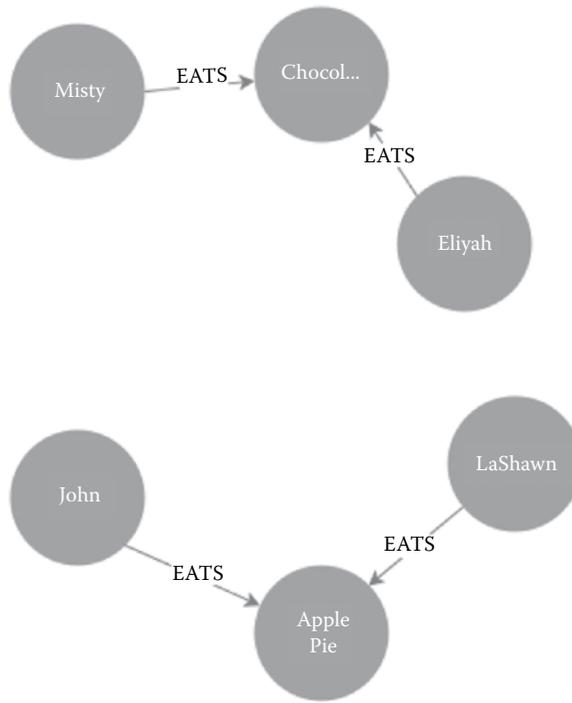


FIGURE 3.7
Nodes with “EATS” relationship.

3.5.4 Importing Specific Rows or Columns

Let us say that you have to give relationship “EATS” between People who eat Apple Pie and Dessert, which in our case would be Apple Pie. [Figure 3.8](#) shows our objective in this section.

We must use the WHERE clause in our query to selective import the CSV. It is important that you read your CSV carefully before importing or else you may end up with an undesired result (Neo4j Developer Manual v3.0, 2016). A point to note here is that WHERE

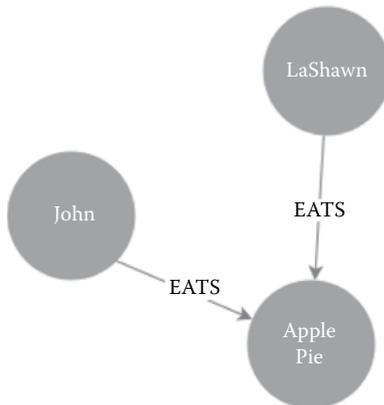


FIGURE 3.8
People who “Eat” Apple Pie.

TABLE 3.20

Exporting to CSV Based on Timestamps

```

SELECT 'D_ID', 'ITEMNAME'
UNION
SELECT * FROM Desserts WHERE DATE(`timestamp_column`) = CURDATE()
INTO OUTFILE 'desserts.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

```

is not a clause in its own right—rather, it is part of `MATCH`, `OPTIONAL MATCH`, `START`, and `WITH`. In our example, we are using `WHERE` coupled with the `MATCH` clause. [Figure 3.8](#) contains the query to selectively add relationship.

3.5.5 Use Case for Joining Two Tables

In Section 3.5.3, we have covered the relationship `EATS` which links the `Dessert` nodes with `Person`. It gives a clear procedure for you to import them and then join them separately using a `MATCH` clause. Therefore, we will not be covering it in this section.

3.5.6 Incremental Imports

As such, there is no such thing as incremental imports in Neo4j. However, there is a roundabout way of including only the newer entries based on your criteria. For now, let us assume that your criteria are based on timestamp. The procedure is to generate a CSV from MySQL as per your timestamp. [Table 3.20](#) exports CSV based on timestamp. The query will return the current day's data.

Then we follow the same procedure as explained in Section 3.5.3.

3.5.7 Conclusion

Cypher Query Language makes it simple for us to import data using `LOAD CSV` to Neo4j. Before importing the CSV, we need to make sure that we have validated the CSV or else our nodes in Neo4j may be inconsistent. Selective importing helps us in improving the performance of the entire process. We know that `PERIODIC COMMIT` helps in effective memory utilization when importing huge CSV files.

3.6 Data Migration: Current and Future Trends

3.6.1 Overview of Google Cloud Dataflow for Pipelining Data across Storage Systems

So far, in this chapter, we have come across various tools for migration of data from RDBMS systems to NoSQL systems. Considering that the theme of this book is NoSQL databases for storage and retrieval of data in the Cloud, it would make this chapter incomplete to not mention some of the latest advances in cloud storage and database technologies that allow for migration of data to and from the cloud storage system to other storage systems like RDMBS, etc.

One such technology is Google’s Cloud Dataflow (Google Cloud Platform—Dataflow, n.d.). Using Cloud Dataflow, we can define data sources and sinks to enable pipelining of data from one source to a sink, which can be of different structures. For example, if we were to use Google’s BigQuery (Google Cloud Platform—BigQuery, n.d.) platform to perform high-speed querying operations on data, which is initially stored in an external source from a Google BigQuery table, such as CSV text files, Avro (Apache Avro™ 1.8.1 Documentation, 2016) Files, or other BigQuery tables, we can make use of Google Cloud Dataflow as a tool to perform this data pipelining operation. The Dataflow SDK supplied by Google allows developers to extend the capabilities of the system by defining custom sources and sinks apart from those supported natively by Cloud Dataflow, and as a result, it is a flexible platform for performing a variety of operations on data that are stored across systems of varying nature.

3.6.2 Other Independent Tools or Solutions for Data Migration

While we have covered a variety of tools for data migration in this chapter, this by no means is an exhaustive listing of data migration tools and is only to be treated as an overview of the approaches that can be taken for data migration from RDBMS systems to NoSQL-based systems. There are many independent developers who have released open-sourced tools for data migration between different database systems implemented in different languages following different practices for migration. Due to lack of a standardized approach to data migration, developers often tend to use custom in-house scripts for handling specific data migration jobs, where the script is configured for only a certain type of data and for migration between fixed set of systems. However, attempts at one tool for all databases have been made, and one such attempt is exemplified by an independently developed tool titled “SQLToNoSQLImporter” (Msathis, 2015) written to handle importing of data from RDBMS (MySQL, Oracle SQL, PostgreSQL) to NoSQL Systems (MongoDB, CouchDB, Elastic Search). The tool uses a lot of configuration files to determine the data migration process including de-normalized schemas, field information, etc. Apache Solr’s Data Import Handler inspires the tool in working, and as a result, a lot of ideas used in the latter are based on those implemented in the former.

Apart from open-source tools, there are services offered by enterprises for performing data migration operations such as Techgene’s Pelica (Techgene, 2016) Migration Tool for MongoDB or DataStax Enterprise (Datastax Enterprise, 2016) for Cassandra which offers easy migration of RDBMS data built-in to their service.

3.7 Practice Exercises

- a. Use *Mongify* to do the following SQL to NoSQL data migration (Figure 3.9).
- b. Make a list of online tools to convert SQL queries to NoSQL database queries and try them out.
- c. Create a database to store user posts, their likes, comments, and comment likes.
- d. Give interpretation of how to tackle the following case study (Figure 3.10).

Person

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

Car

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100,000	0
102	Rolls Royce	1965	330,000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150,000	4
105	Ranault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

FIGURE 3.9 MongoDB document creation using Mongify.

P. K. Hallinan

A Rainbow of Friends



Image Source: Cover Page of Book 'A Rainbow of Friends Paperback' Dec, 2005.

Example - Social Network Graph:
Each record: UserID1, UserID2
 Separate records: UserID, first_name, last_name, age, gender, ...
Task: Find all teenage female friends of friends of friends of ... friends of a given user, who likes horror movies, the colour blue and 'croissant'.
Challenge:

- a) The data 'teenage' (derived from age) and 'female' (derived from gender) can be added or removed to the records dynamically.
- b) The data such as the 'likes' of friends are not formatted and stored in a table schema-like structure.
- c) The application data is inherently distributed across multiple networked servers in data centers across the globe.
- d) To handle increasing load on the application, the deployment platform is upgraded by adding more servers, to the existing ones, rather than buying better higher-end servers and migrating the data to these new servers.
- e) High availability of information to the application users is of great importance, even at the cost of data inconsistency.

FIGURE 3.10 Friends—social network graph.

- e. The URL (<https://www3.ntu.edu.sg/home/ehchua/programming/sql/SampleDatabases.html>) contains several DB schemas along with the data. The databases have more than a million records. However, the simplest one is the “Employee” database, which has six tables. Go ahead, and try to out migration to NoSQL databases for bigger data sets.
- f. There is a Visual Modeling tool available at <http://www.dbschema.com/mongodb-tool.html> which automatically converts the MongoDB JSON documents created into graphical/visual form. Try it out for the NoSQL you experimented with.
- g. The above Visual Modeling tool is a paid one, with a trial period of 5 days. Find out equivalent freeware or student edition tools.

References

- Anlek. 2016. Anlek/Mongify. Retrieved March 06, 2016, from <https://github.com/anlek/mongify>
- Apache Avro™ 1.8.1 Documentation. 2016. Retrieved May 30, 2016, from <http://avro.apache.org/docs/current/>
- Apache Hadoop 2.7.2. 2016. Retrieved May 19, 2016, from <https://hadoop.apache.org/docs/r2.7.2/>
- Apache Hive™. 2014. Retrieved April 22, 2016, from <https://hive.apache.org>
- Beaumont, D. 2014. How to Explain Horizontal and Vertical Scaling. Retrieved March 25, 2016, from <http://www.thoughtsoncloud.com/2014/04/explain-vertical-horizontal-scaling-cloud/>
- Big Data. Retrieved March 14, 2016, from <https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>
- Cogswell, J. 2013. Why My Team Went with DynamoDB over MongoDB. Retrieved May 11, 2016, from <http://insights.dice.com/2013/02/21/why-my-team-went-with-dynamodb-over-mongodb/>
- Datastax—eBay. 2014. Retrieved May 12, 2016, from <http://www.datastax.com/wp-content/uploads/2012/12/DataStax-CS-eBay.pdf>
- Datastax Enterprise. 2016. Retrieved June 1, 2016, from <http://www.datastax.com/products/datastax-enterprise>
- Google Cloud Platform—BigQuery. n.d.. Retrieved May 28, 2016, from <https://cloud.google.com/bigquery/>
- Google Cloud Platform—DataFlow. n.d.. Retrieved May 27, 2016, from <https://cloud.google.com/dataflow/>
- JavaSE Downloads. 2016. Retrieved March 18, 2016, from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Keep, M. 2014. Migrating from Relational Databases to MongoDB. Retrieved March 17, 2016, from <http://www.slideshare.net/matkeep/migrating-from-relational-databases-to-mongodb>
- Msathis. 2015. Msathis/SQLToNoSQLImporter. Retrieved June 08, 2016, from <https://github.com/msathis/SQLToNoSQLImporter>
- MySQL Foodmart DB. 2008. Retrieved March 20, 2016, from <http://pentaho.dlpage.phi-integration.com/mondrian/mysql-foodmart-database>
- Neo4j Developer Manual v3.0. 2016. Retrieved November 6, 2016, from <http://neo4j.com/docs/developer-manual/current/cypher/#query-periodic-commit>
- Neo4j at Walmart. 2016. Retrieved May 10, 2016, from <http://neo4j.com/case-studies/walmart/>
- Rocha, Á., Correia, A. M., Adeli, H., Reis, L. P., and Mendonça Teixeira, M. Eds. 2016. *New Advances in Information Systems and Technologies*. Volume 1. Retrieved March 14, 2016, from <https://books.google.co.in/books?isbn=3319312324>
- Sqoop Userguide V1.4.6. 2016. Retrieved June 08, 2016, from <https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html>

- Techgene. 2016. Retrieved June 4, 2016, from <http://www.techgene.com/it-solutions/data-migration/>
- Trivedi, A. 2014. Mapping Relational Databases and SQL to MongoDB. Retrieved April 04, 2016, from <http://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650>
- What is a Graph Database?. (n.d.). Retrieved November 6, 2016, from <https://neo4j.com/developer/graph-database/#property>



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

4

Comparative Study on Mostly Used NoSQL Databases

K. G. Srinivasa, Nabeel Siddiqui, and Abhishek Kumar

CONTENTS

4.1	Introduction.....	75
4.2	NoSQL Databases	76
4.2.1	SimpleDB.....	76
4.2.1.1	Data Model.....	77
4.2.1.2	Consistency	79
4.2.1.3	Limitation of SimpleDB.....	79
4.2.2	Google's BigTable	80
4.2.2.1	Data Model.....	80
4.2.2.2	Sharding	82
4.2.2.3	Replication and Failure Handling	82
4.2.3	MongoDB	83
4.2.3.1	Data Model.....	83
4.2.3.2	Sharding	85
4.2.3.3	Architecture	87
4.2.3.4	Consistency	88
4.2.3.5	Replication and Failure Handling	89
4.2.4	CouchDb.....	90
4.2.4.1	Data Model.....	90
4.2.4.2	CouchDB Architecture	90
4.2.4.3	Sharding	92
4.2.4.4	Consistency	94
4.2.4.5	Replication and Failure Handling	96
	References.....	97

4.1 Introduction

This chapter includes a selection of NoSQL databases that have different operational capabilities and function. The large, complex, heterogeneous, structured or unstructured data under the scope of Big Data have attracted attention and momentum in past few years. The pandemic usage of social media, such as Facebook and Twitter, is majorly the culprit for such dynamic proliferation. The volume expansion of Big Data with such a fast pace imposes big research challenges. The data size has experienced a growth from few terabytes to petabytes in one single data set. It is a general observation that the expansion

of data has seen more growth than the impending management techniques required to tackle such huge data set, and this results in the paucity of robust and appropriate data engineering systems. Most of the data processing systems still rely on their legacies of 1980s and 1990s, despite the significant recurring remodeling endeavors. The timely reviews on computing solutions for handling the mammoth high-dimensional data sets, especially in healthcare, offer some early insights of impending Big Data era. The reviews suggest that the existing technologies for years could be the potential solutions, but question about their computing capacities are unanswered. As their usages not only increase additional complexity, but also they could not sustain the stress of a large amount of data, travels between the computing nodes often break down. These parameters are intrinsically important in deciding when to use a particular NoSQL database.

This chapter focuses on the classification of different NoSQL databases. The technical comparison strategy has been decided based on the parameters on which these databases are measured. For a better comparison, the databases in this chapter are analyzed with several aspects in mind. The first one is the Data Model, which defines how a database stores data and how it handles concurrent access. The second one is the Query Model, in which the surveyed databases differ the most. This category contains the power of the used query language, its restrictions, and how it can be used. The next two examined facets that determine how the databases can be distributed to other many machines are Sharding and Replication. And, in the Consistency category, the consistency level the databases achieve and tradeoffs they make are analyzed.

Some of the technical details of the implementations are taken care of in the Architecture sections. Another important aspect is how the databases can handle various types of failures, like single-node failures and network problems; this is discussed under the term Failure Handling.

4.2 NoSQL Databases

4.2.1 SimpleDB

Amazon's SimpleDB is a highly ranked NoSQL data store that reduces the intricacies of administering database. It is a web service that provides you core database features like fast lookup of the data, querying search results, etc. Unlike the stringent requirements of a relational database, Amazon SimpleDB is optimized to provide high availability and flexibility, with no administrative burden. SimpleDB generates and handles multiple geographically distributed replicas of your data automatically to enable greater availability and data durability.

Amazon's SimpleDB is considered as the most valuable storage solution for building applications from the scratch since it routinely carries out many day-to-day jobs associated with management and scalability of database. It removes the demand on the Developers to consider data modeling, maintaining of index and optimizing performance (tuning), or data manipulation (also done automatically). It is a perfect solution for existing and maintainable applications. Existing applications experience a new life with the impeccable data storage of Amazon's SimpleDB. We can also refactor huge applications using cloud-based data store, thereby increasing the scalability, minimizing operational expenses, and considerably reducing database administration time [1].

4.2.1.1 Data Model

The Simple DB data model primarily comprises the following four parts: domains, items, attributes, and values. The hierarchal concept of these is shown in Figure 4.1.

A *domain* is a container that lets you reserve your structured data and run queries on it. The data are stored in the domain as items. As an example, the domain can be thought of as a worksheet tab in spreadsheet and items are like rows of the spreadsheet. Queries can be run against the domains, but it cannot be run across domains of current version. Each domain in your SimpleDB account is distinctly different from all your other domains, and therefore, the items stored in one domain are completely separate from the items stored in other domains. This is the reason why queries cannot be performed across domains. You can store all of your data in a single domain or partition it across multiple domains, depending on the nature of the data and the application. You can create a domain called bikes and use it, or you can partition the data into separate domains such as brands beginning with A-J in BIKESAJ, beginning with K-T in BIKESKT, and so on. SimpleDB gives you the freedom to partition the data however you like.

4.2.1.1.1 SimpleDB Domain Constraints

You should also be familiar with some of the constraints and limitations when using the SimpleDB domains.

Number of domains: Each SimpleDB account is by default allowed to create up to 100 domains. If your architecture or system design should need more than 100 domains, you can request AWS to increase the limit. Normally, it takes about two business days for the request to be approved. You need to fill out a form on the SimpleDB site (<http://aws.amazon.com/contact-us/simpledb-limit-request/>) to submit your request.

Size of domain: The size of a single domain is limited to 10 GB of data. This includes all the storage used by the data for the domain. This is a hard limit, and currently, there is no way to request for an increase.

Number of attributes: The total number of attributes stored in a domain is limited to one billion per domain. That is a lot of attributes and most applications should comfortably fit within this limit. This is considered per domain, so with the default number

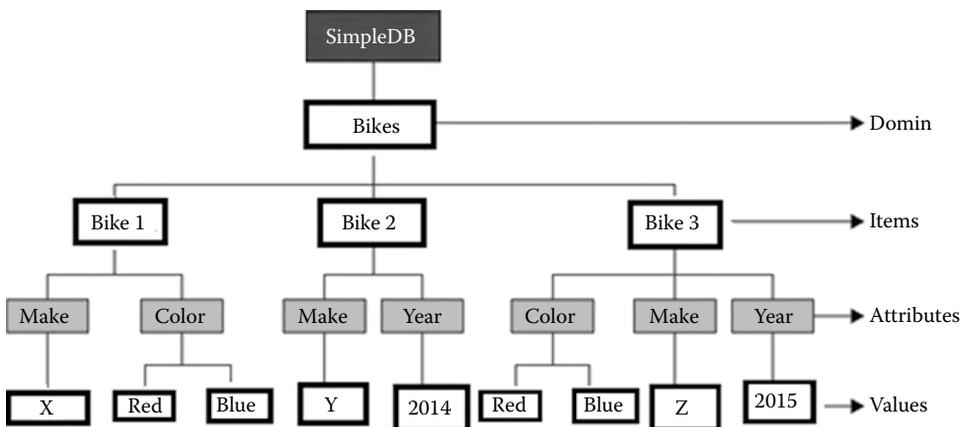


FIGURE 4.1 Hierarchal concept of SimpleDB.

of 100 domains, you are looking at a total allowed limit of 100 billion attributes! Partitioning data across multiple domains can be used to get around the limit.

Name of a domain: The name of a domain must be at least three characters in length and a maximum of 255 characters. The allowed characters in the name are a–z, A–Z, 0–9, *_*, *.*, and *-*.

4.2.1.1.2 Items

Items comprise individual objects within each domain, and each item contains attributes with values. As explained above, items represent rows in spreadsheet. Items are marked with a unique ID, which can be used for debugging and querying for specific items. The unique identifier needs to be provided by the developer and not by SimpleDB.

4.2.1.1.3 Generation of Unique IDs by Developer

Identifiers or *GUIDs* are used as a key for each item. This scheme is not mandatory to be used by the developer, and an ID can be generated as far as it is unique. The ID also needs to be unique within a domain of interest. You can add attributes to an item in SimpleDB. Any items that you create that do not have any attributes will not be returned in queries. SimpleDB treats empty items as nonexistent.

4.2.1.1.4 Constraints on SimpleDB Items

The constraints that you need to be aware of when working with SimpleDB items are as follows:

- The length of the name for an item cannot be more than 1024 bytes.
- Each item can have a maximum of 256 name–value pairs per item in a domain.
- The name of an item must only use characters that are UTF-8 characters, which are valid in XML documents. Control characters and any sequences that are not valid in XML are not allowed for use as part of an item name.

4.2.1.1.5 Attributes

Each item will have attributes, similar to the concept of spreadsheet or a column in a table of a database. Attributes are represented by key–value pair. Key represents the unique name of the attribute, and value represents the textual data. SimpleDB being schema less allows different attributes for each item in a domain. This is impossible in a relational database world where you must define your table schemas up front, and every time you need to add a new field or column, you must upgrade the schema for the database, or your existing applications might start throwing errors. SimpleDB frees the user from this constraint of upgrade and maintenance cycle, and it gives the flexibility to design our own application. If an attribute is added to an item in a domain, only that item will have that attribute, and all the other existing items in the domain will play along nicely without that additional attribute.

4.2.1.1.6 Constraints on SimpleDB Item Attributes

Simple DB item attributes have the following constraints:

- The length of the name for an attribute cannot be more than 1024 bytes.
- The name of an attribute must only use characters that are UTF-8 characters, which are valid in XML documents. Control characters and any sequences that are not valid in XML are not allowed for use as part of an attribute name.

4.2.1.1.7 Values

Each attribute is a key–value pair, and data are stored in the value. Only textual data can be stored in SimpleDB. There are some workarounds to store data as binary data in Amazon S3 and use metadata in SimpleDB to point to it. The only restriction textual data has, with a larger implication, is that you must encode and decode values for other data types such as dates and numbers when storing and retrieving them for use in your application. One of the unique features of SimpleDB is the ability to have multiple values for a single attribute. These multiple values are actually stored by SimpleDB in such a way that you can query for each separately.

4.2.1.1.8 Constraints on Values of a SimpleDB Item

A value stored in an attribute must follow the following restrictions:

- The length of the value cannot be more than 1024 bytes.
- The value must only use characters that are UTF-8 characters, which are valid in XML documents. Control characters and any sequences that are not valid in XML are not allowed for use as part of the value. This is true for SOAP requests. Invalid XML characters inserted when using the REST API will be returned as base64-encoded values when retrieving an item later [2].

4.2.1.2 Consistency

Multiple copies of each domain are stored in SimpleDB. A successful write (using PutAttributes, BatchPutAttributes, DeleteAttributes, BatchDeleteAttributes, CreateDomain, or DeleteDomain) guarantees that all copies of the domain will durably persist.

SimpleDB supports consistent read and eventual read explained below.

An eventual read (using Select or GetAttributes) might not reflect the results of a recently completed write (using PutAttributes, BatchPutAttributes, DeleteAttributes, or BatchDeleteAttributes). Consistency of read is usually reached within a short span of time.

A consistent read (using Select or GetAttributes with ConsistentRead=true) returns a result that reflects all writes that received a successful response prior to the read.

By default, GetAttributes and Select perform an eventually consistent read.

Table 4.1 demarcates the difference between the two reads.

4.2.1.3 Limitation of SimpleDB

Table 4.2 describes current limits within Amazon SimpleDB. The table also describes the characteristics of eventually consistent read and consistent read.

TABLE 4.1

Difference between Eventual and Consistent Reads

Eventually Consistent Read	Consistent Read
Stale reads possible	No stale reads
Lowest read latency	Potential higher read latency
Highest read throughput	Potential lower read throughput

TABLE 4.2

Limits of SimpleDB Parameters

Parameter	Restriction
Domain size	10 GB per domain
Domain size	1 billion attributes per domain
Domain name	3–255 characters (a–z, A–Z, 0–9, ‘_’, ‘-’, and ‘.’)
Domains per account	250
Attribute name–value pairs per item	256
Attribute name length	1024 bytes
Attribute value length	1024 bytes
Item name length	1024 bytes
Attribute name, attribute value, and item name allowed characters	All UTF-8 characters that are valid in XML documents Control characters and any sequences that are not valid in XML are returned Base64-encoded. For more information, see Working with XML-Restricted Characters
Attributes per PutAttributes operation	256
Attributes requested per Select operation	256
Items per BatchDeleteAttributes operation	25
Items per BatchPutAttributes operation	25
Maximum items in Select response	2500
Maximum query execution time	5 seconds
Maximum number of unique attributes per Select expression	20
Maximum number of comparisons per Select expression	20
Maximum response size for Select	1 MB

Source: Adapted from Chaganti, P. and Helms, R. 2010. Amazon SimpleDB Developer Guide Scale Your Application’s Database on the Cloud Using Amazon SimpleDB; Fowler, A. n.d. NoSQL for Dummies; Yadava, H. 2007. *The Berkeley DB Book*. Berkeley, CA: Apress.

4.2.2 Google’s BigTable

BigTable is a light, scattered, constant, multidimensional, sorted map database. This map’s indexing is done by row key, column key, and a timestamp. In BigTable, value is determined as array of bytes. BigTable stores structured information. It does not impose any size restrictions for each value. BigTable is highly scalable and can range up to thousands of computers. BigTable is designed on top of Google File System. Chubby is used to store the root tablet, schema details, access control lists, coordinate, and identify tablet servers. The automatic clean-up process of BigTable is done by removing SST tables and unused data by using Mark and Sweep algorithm.

4.2.2.1 Data Model

The Data Model of BigTable is perceived by an example to make it understandable. Let us use table “URLtable” for example (depicted in Figure 4.2). In URLtable, we would use URLs as row keys, various aspects of web pages as column names, and store the contents of the web pages in the contents: column under the timestamps when they were fetched, as illustrated in Figure 4.2.

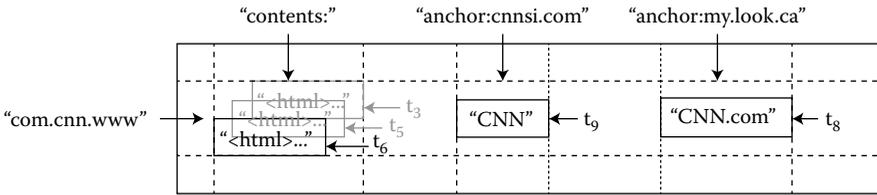


FIGURE 4.2
URLtable example.

4.2.2.1.1 Rows

The row keys in a table are arbitrary strings (currently up to 64 KB in size, although 10–100 bytes is a typical size for most of our users). Every read or write of data under a single row key is atomic, and a design decision makes it easier for clients to reason about the system’s behavior in the presence of concurrent updates to the same row. BigTable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing. As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines. Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses. For example, in URLtable, pages in the same domain are grouped together into contiguous rows by reversing the hostname components of the URLs. For example, we store data for maps.google.com/index.html under the key com.google.maps/index.html. Storing pages from the same domain near each other makes some host and domain analyses more efficient.

4.2.2.1.2 Column

Families Column keys are grouped into sets called column families, which form the basic unit of access control. All data stored in a column family are usually of the same type (we compress data in the same column family together). A column family must be created before data can be stored under any column key in that family; after a family has been created, any column key within the family can be used. It is our intent that the number of distinct column families in a table be small (in the hundreds at most) and that families rarely change during operation. In contrast, a table may have an unbounded number of columns. Column family names must be printable, but qualifiers may be arbitrary strings. An example column family for the URLtable is language, which stores the language in which a web page was written. We use only one column key in the language family, and it stores each web page’s language ID. Another useful column family for this table is anchor; each column key in this family represents a single anchor, as shown in Figure 4.2. The qualifier is the name of the referring site; the cell content is the link text. Access control and both disk and memory accounting are performed at the column-family level. In our URLtable example, these controls allow us to manage several different types of applications: some that add new base data, some that read the base data and create derived column families, and some that are only allowed to view existing data (and possibly not even to view all of the existing families for privacy reasons).

4.2.2.1.3 Timestamps

Each cell in a BigTable can contain multiple versions of the same data; these versions are indexed by timestamp. BigTable timestamps are 64-bit integers. They can be assigned

by BigTable, in which case they represent “real time” in microseconds or be explicitly assigned by client applications. Applications that need to avoid collisions must generate unique timestamps themselves. Different versions of a cell are stored in decreasing timestamp order so that the most recent versions can be read first. To make the management of versioned data less onerous, we support two per-column-family settings that tell BigTable to garbage-collect cell versions automatically. The client can specify either that only the last n versions of a cell be kept or that only new-enough versions be kept. In our URLtable example, we set the timestamps of the crawled pages stored in the contents:column to the times at which these page versions were actually crawled. The garbage-collection mechanism described above lets us keep only the most recent three versions of every page.

4.2.2.2 Sharding

Database Sharding is a popular topic over the past several years. Due to high volume of transaction data and enormous growth in application data, Sharding is an important to-do list for every designer [1].

Database Sharding can be simply defined as a “shared-nothing” partitioning scheme for large databases across a number of servers, enabling new levels of database performance and scalability achievable. If you think of broken glass, you can get the concept of sharding—breaking your database down into smaller chunks called “shards” and spreading those across a number of distributed servers [6].

4.2.2.3 Replication and Failure Handling

4.2.2.3.1 Reliable Storage: Durability and Replication

Mostly, any storage system aims to store data *reliably* so that if a computer fails, the data can be recovered. We worry about both *temporary* failures, where a computer goes offline for a while but will come back, and *permanent* failures, where a computer dies. Network partitions, power blips, and program crashes generally cause temporary failures; hardware failure, fires, and sabotage generally cause permanent failures. We assume (with good reason) that temporary failures are more common and unpredictable than permanent ones.

To guard against power blips and program crashes, a system must store data on *durable* media, such as disks and flash memory. Only data stored on durable media will survive reboot. (Reboot is a magic solution for many temporary failures.)

But, durable media cannot guard against permanent failures. That requires *replication*, where the system keeps multiple copies of the data: backups, basically. If the data are stored several times on several geographically distributed computers, then only a major catastrophe will cause data loss.

Most (but, interestingly, not all) distributed systems use *both* durability *and* replication to store data reliably. For instance, each data modification might be written to at least three disks. If one disk fails, the data are proactively copied onto a new disk so that at least three copies are usually available. That way, only three simultaneous permanent failures cause data loss. (Nondurable replication has not been considered sufficient since temporary failures—which are more common, and so might happen simultaneously—lose nondurable data.)

Most GFS files are replicated to three computers, which write them durably onto disks and flash.

4.2.2.3.2 Sequential Storage

GFS was designed to store very large files that are generally accessed sequentially: starting from the first byte and proceeding in order. Sequential access is almost always the fastest way to access files on any storage system. Why?

- Because hard disks are mechanical objects that spin. Reading data in a random order asks a disk's mechanical "head" to jump around, a process called *seeking*. The head estimates the place to jump to and then must settle to get it right. A disk can do at most a couple *hundred* seeks a second. Sequential access (on sequentially laid out files) avoids seeking. Although in flash memory the seek penalty is much smaller, it still exists.
- Because sequential access is predictable, all system caches have an easier job. The operating system can prefetch future data, dramatically speeding up future reads, simply by reading the next couple blocks of the file. The disk/flash itself can do the same thing with on-drive caches.

4.2.2.3.3 Structured Storage

BigTable, however, stores structured data, including large items (like web pages) and small items (like the text of a link). A typical BigTable transaction might involve only a couple of small data items, but many clients may access a BigTable at a time. This offers both performance and correctness challenges. How can such a system scale?

BigTable makes a couple of data model choices relevant for our understanding:

- *Sparse hashtable*. A BigTable is essentially a sparse 3D hash table, where the dimensions are row names, column names, and versions (timestamps).
- *Strings*. All BigTable row names, column names, and data items are strings (sequences of characters). BigTable has no true schema: everything is a string.
- *Put, get, scan*. BigTable supports four fundamental operations: *put* (store a value in a row/column entry), *get* (return the value in a row/column entry), *delete* (delete a row/column entry), and *scan* (return many values from many row/column entries, in sorted order) [7].

4.2.3 MongoDB

4.2.3.1 Data Model

Unlike other databases where you must determine schema before entering data, MongoDB has a flexible schema. The flexibility allows mapping of documents to an entity or an object. The key challenge in data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns.

4.2.3.1.1 Document Structure

An important aspect of designing data models is to formulate relationships between data. There are specifically two tools that allow applications to represent these. They are explained in the following.

4.2.3.1.2 References

References store the relationships between data across documents by links or references. These are often referred to as normalized data models (Figure 4.3).

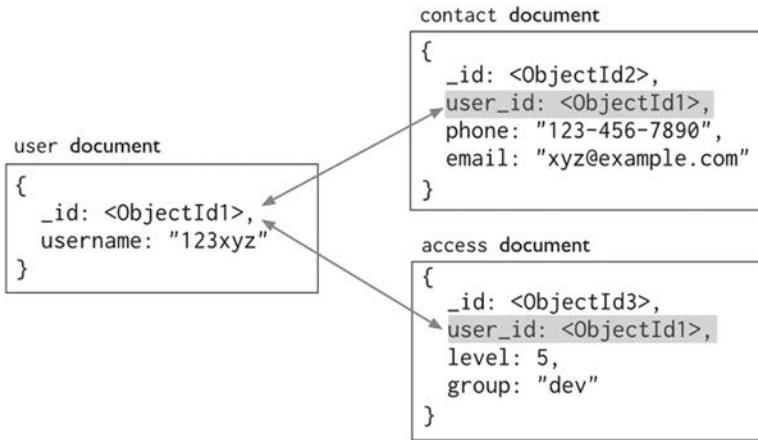


FIGURE 4.3
Normalized data models for the strengths and weaknesses of using references.

4.2.3.1.3 *Embedded Data*

Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These *denormalized* data models allow applications to retrieve and manipulate related data in a single database operation (Figure 4.4).

4.2.3.1.4 *Atomicity of Write Operations*

In MongoDB, write operations are atomic at the document level, and no single write operation can atomically affect more than one document or more than one collection. A denormalized data model with embedded data combines all related data for a represented entity in a single document. This facilitates atomic write operations since a single write operation can insert or update the data for an entity. Normalizing the data would



FIGURE 4.4
Embedded data models for the strengths and weaknesses of embedding documents.

split the data across multiple collections and would require multiple write operations that are not atomic collectively.

However, schemas that facilitate atomic writes may limit ways that applications can use the data or may limit ways to modify applications. The Atomicity Considerations documentation describes the challenge of designing a schema that balances flexibility and atomicity.

4.2.3.1.5 Document Growth

Some updates, such as pushing elements to an array or adding new fields, increase a document's size.

For the MMAPv1 storage engine, if the document size exceeds the allocated space for that document, MongoDB relocates the document on disk. When using the MMAPv1 storage engine, growth consideration can affect the decision to normalize or denormalize data. See Document Growth Considerations for more about planning for and managing document growth for MMAPv1.

4.2.3.1.6 Data Use and Performance

When designing a data model, consider how applications will use your database. For instance, if your application only uses recently inserted documents, consider using Capped Collections. Or, if your application needs are mainly read operations to a collection, adding indexes to support common queries can improve performance.

See Operational Factors and Data Models for more information on these and other operational considerations that affect data model designs [8].

4.2.3.2 Sharding

Sharding is supported by MongoDB through configuration of shared clusters. Shared clusters have the following components:

Shards store the data. To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set.

Query routers, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. A client sends requests to a mongos, which then routes the operations to the shards and returns the results to the clients. A sharded cluster can contain more than one mongos to divide the client request load, and most sharded clusters have more than one mongos for this reason.

Configuration servers store the cluster's metadata. These data contain a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards (Figure 4.5).

4.2.3.2.1 Data Partitioning

MongoDB distributes data, or shards, at the collection level. Sharding partitions a collection's data by the *shard key*.

4.2.3.2.2 Shard Keys

To shard a collection, you need to select a *shard key*. A shard key is either an indexed field or an indexed compound field that exists in every document in the collection. MongoDB divides the shard key values into *chunks* and distributes the chunks evenly across the

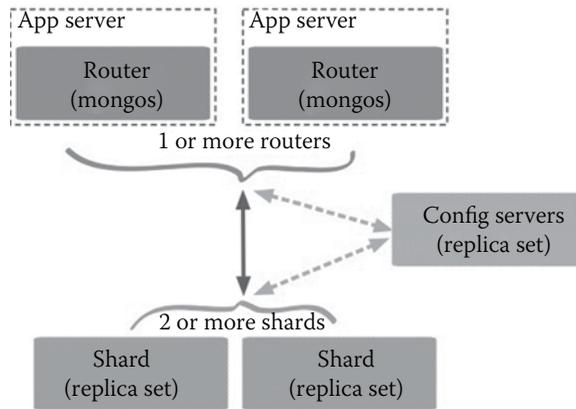


FIGURE 4.5 Sharding in MongoDB.

shards. To divide the shard key values into chunks, MongoDB uses either *range-based partitioning* or *hash-based partitioning*.

4.2.3.2.3 Range-Based Sharding

For *range-based sharding*, MongoDB divides the data set into ranges determined by the shard key values to provide *range-based partitioning*. Consider a numeric shard key: If you visualize a number line that goes from negative infinity to positive infinity, each value of the shard key falls at some point on that line. MongoDB partitions this line into smaller, nonoverlapping ranges called *chunks* where a chunk is a range of values from some minimum value to some maximum value.

Given a range-based partitioning system, documents with “close” shard key values are likely to be in the same chunk, and therefore on the same shard (Figure 4.6).

4.2.3.2.4 Hash-Based Sharding

For *hash-based partitioning*, MongoDB computes a hash of a field’s value and then uses these hashes to create chunks.

With hash-based partitioning, two documents with “close” shard key values are *unlikely* to be part of the same chunk. This ensures a more random distribution of a collection in the cluster (Figure 4.7).

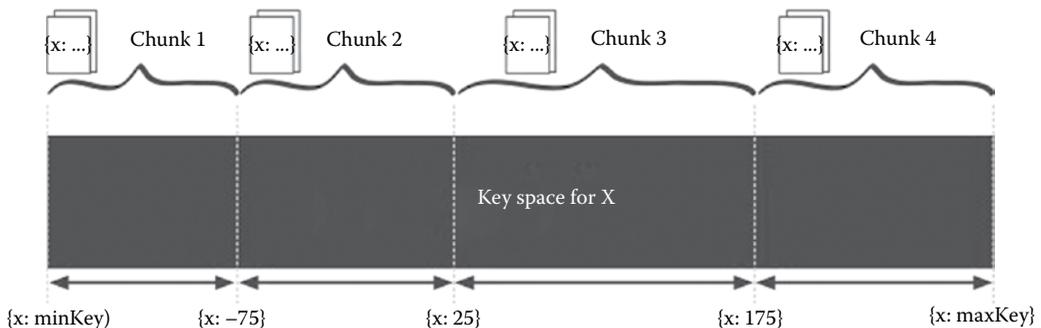


FIGURE 4.6 Depiction of range-based sharding.

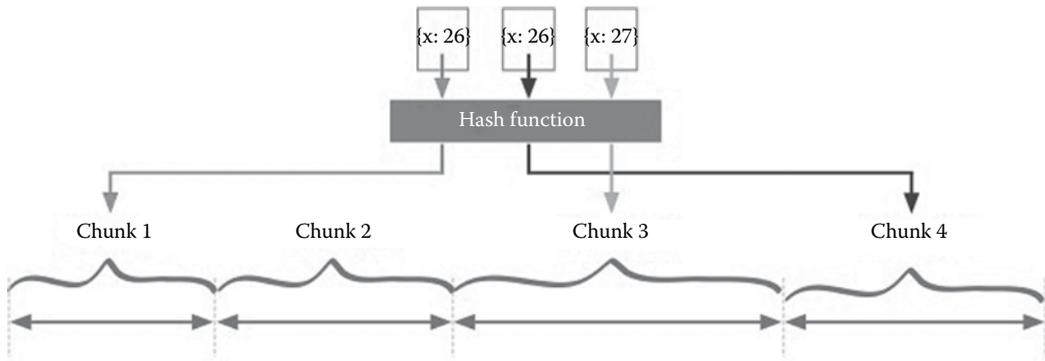


FIGURE 4.7
Depiction of hash-based sharding.

4.2.3.3 Architecture

MongoDB’s design philosophy is focused on combining the critical capabilities of relational databases with the innovations of NoSQL technologies. Our vision is to leverage the work that Oracle and others have done over the past four decades to make relational databases what they are today. Rather than discarding decades of proven database maturity, MongoDB is picking up where they left off by combining key relational database capabilities with the work that Internet pioneers have done to address the requirements of modern applications (Figure 4.8).

Relational databases have reliably served applications for many years and offer features that remain critical today as developers build the next generation of applications:

- *Expressive query language and secondary indexes.* Users should be able to access and manipulate their data in sophisticated ways to support both operational and analytical applications. Indexes play a critical role in providing efficient access to data, supported natively by the database rather than maintained in application code.

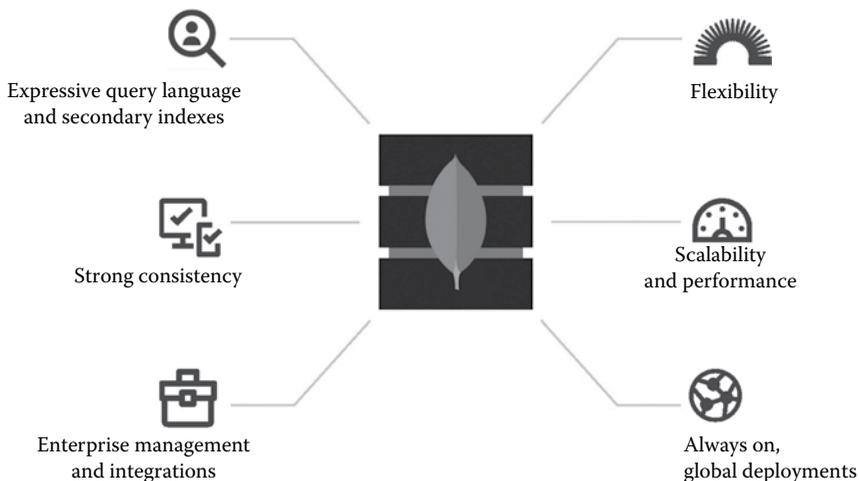


FIGURE 4.8
Architecture of MongoDB.

- *Strong consistency.* Applications should be able to immediately read what has been written to the database. It is much more complex to build applications around an eventually consistent model, imposing significant work on the developer, even for the most sophisticated engineering teams.
- *Enterprise management and integrations.* Databases are just one piece of application infrastructure and need to fit seamlessly into the enterprise IT stack. Organizations need a database that can be secured, monitored, automated, and integrated with their existing technology infrastructure, processes, and staff, including operations teams, DBAs, and data analysts.

However, modern applications impose requirements not addressed by relational databases, and this has driven the development of NoSQL databases, which offer the following:

- *Flexible data model.* NoSQL databases emerged to address the requirements for the data we see dominating modern applications. Whether document, graph, key-value, or wide-column, all of them offer a flexible data model, making it easy to store and combine data of any structure and allow dynamic modification of the schema without downtime or performance impact.
- *Scalability and performance.* NoSQL databases were all built with a focus on scalability, so they all include some form of sharding or partitioning. This allows the database to scale out on commodity hardware deployed on-premises or in the cloud, enabling almost unlimited growth with higher throughput and lower latency than relational databases.
- *Always-on global deployments.* NoSQL databases are designed for highly available systems that provide a consistent, high-quality experience for users all over the world. They are designed to run across many nodes, including replication to automatically synchronize data across servers, racks, and data centers.

While offering these innovations, NoSQL systems have sacrificed the critical capabilities that people have come to expect and rely on from relational databases. MongoDB offers a different approach. With its Nexus Architecture, MongoDB is the only database that harnesses the innovations of NoSQL while maintaining the foundation of relational databases [8,9].

4.2.3.4 Consistency

4.2.3.4.1 Monotonic Reads

MongoDB provides monotonic reads from a standalone mongod instance. Suppose an application performs a sequence of operations that consists of a read operation R_1 followed later in the sequence by another read operation R_2 . If the application performs the sequence on a standalone mongod instance, the later read R_2 never returns results that reflect an earlier state than that returned from R_1 ; that is, R_2 returns data that are monotonically increasing in recency from R_1 .

Changed in version 3.2: For replica sets and sharded clusters, MongoDB provides monotonic reads if read operations specify Read Concern “majority” and read preference primary.

In previous versions, MongoDB cannot make monotonic read guarantees from replica sets and sharded clusters.

4.2.3.4.2 Monotonic Writes

MongoDB provides monotonic write guarantees for standalone mongod instances, replica sets, and sharded clusters.

Suppose an application performs a sequence of operations that consists of a write operation W_1 followed later in the sequence by a write operation W_2 . MongoDB guarantees that W_1 operation precedes W_2 .

4.2.3.5 Replication and Failure Handling

MongoDB handles replication through an implementation called “replication sets.” Replication sets in their basic form are somewhat similar to nodes in a master–slave configuration. A single primary member is used as the base for applying changes to secondary members.

The difference between a replication set and master–slave replication is that a replication set has an intrinsic automatic failover mechanism in case the primary member becomes unavailable.

- *Primary member:* The primary member is the default access point for transactions with the replication set. It is the only member that can accept write operations.

Each replication set can have only one primary member at a time. This is because replication happens by copying the primary’s “oplog” (operations log) and repeating the changes on the secondary’s data set. Multiple primaries accepting write operations would lead to data conflicts.

- *Secondary members:* A replication set can contain multiple secondary members. Secondary members reproduce changes from the oplog on their own data.

Although by default applications will query the primary member for both read and write operations, you can configure your setup to read from one or more of the secondary members. A secondary member can become the primary if the primary goes offline or steps down.

- *Arbiter:* An arbiter is an optional member of a replication set that does not take part in the actual replication process. It is added to the replication set to participate in only a single, limited function: to act as a tiebreaker in elections.

In the event that the primary member becomes unavailable, an automated election process happens among the secondary nodes to choose a new primary. If the secondary member pool contains an even number of nodes, this could result in an inability to elect a new primary due to a voting impasse. The arbiter votes in these situations to ensure that a decision is reached.

If a replication set has only one secondary member, an arbiter is required [10].

4.2.4 CouchDb

4.2.4.1 Data Model

4.2.4.1.1 Data Organization

This section describes the fundamental data model characteristics that CouchDB provides.

1. *Data model:* CouchDB implements a Document data model.
2. *Fixed schema:* Many NOSQL databases do not enforce a fixed schema definition for the data store in the database. This enables changes in data structures to be smoothly evolved at the database level over time, enhancing modifiability. Typically, objects with different formats can be stored together in the same collections. In CouchDB, a fixed schema definition is not required.
3. *Opaque data objects:* A database may store data as opaque objects that require interpretation in the client that issues a query. Opaque data objects usually require embedded metadata for interpretation stored with every object, and this may lead to “size bloat” in a database, negatively impacting performance. In CouchDB, opaque data objects are not required.
4. *Hierarchical data objects:* A database support many hierarchical data structures, often known as sub-objects or documents. Graph databases support hierarchical databases naturally in their data model. In CouchDB, sub-objects are supported [11].

4.2.4.2 CouchDB Architecture

4.2.4.2.1 Peers

This document describes ShiftSpace’s use of CouchDB in particular how peers interact and how data are replicated between them. There is no real difference between the ShiftServer that runs on a remote server and the ShiftServer that runs on the client. However, ShiftServer can be configured to behave differently in order to support the querying of shared user content as well as handling the distribution of such shared content to authenticated parties (Figure 4.9).

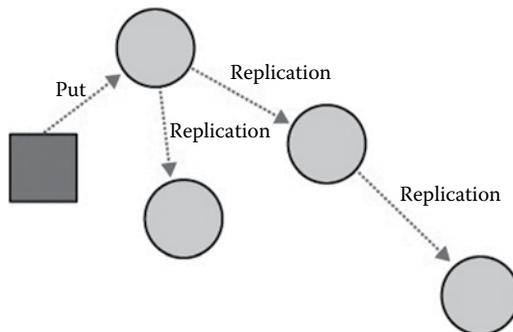


FIGURE 4.9
CouchDB architecture.

4.2.4.2.2 Shifts

The central document to ShiftSpace is the Shift. A Shift is simply a piece of JSON data that a Space (a ShiftSpace extension application) can use to recreate a modification to a website.

When a user makes a modification to a website with a Space, a shift document is saved locally on the user's machine. It is not sent to the server.

4.2.4.2.3 Local Databases

When a user creates an account, five databases (three local, two remote) are created. The local ones are described here.

4.2.4.2.3.1 User/Private This holds all of the user's unpublished shifts. This is replicated only to the local *user/feed*. *user/private* is never replicated to a peer or a publishing server. Thus, there are at least two local copies of a private unpublished shift. There is an additional remote copy on the publishing server for each user or group that the shift gets published to. Sending shifts directly to a user does not scale well because multiple copies must exist in each remote *user/inbox* that the shift is published to. Publishing to a group is much more efficient. Like Twitter, we imagine that most people will make their shifts public or publish them to groups. In all likelihood, only a small majority of power users will use the combination of sending to many individual users and groups

4.2.4.2.3.2 User/Public Whenever a user makes a shift public, it is copied to *user/public* and deleted from *user/private*. This *user/public* is then replicated to *user/feed*. If the user is connected to a peer running as a publishing server, it is copied to the master public database on that server. A shift in *user/public* has at least four copies, two local and two remote (*remote/master* and *remote/shared*).

4.2.4.2.3.3 User/Feed This is the user's view of the world. If the user is running ShiftSpace in full peer mode (replicating the public firehose onto their local machine), queries only happen against *user/feed*. If the user is not running in full peer mode, queries happen against *remote/shared*. The local databases *user/public* and *user/private* are replicated into *user/feed*. In addition, the remote databases *remote/master*, any of the user's groups, *user/inbox*, *user/messages*, and *system/messages* are replicated into *user/feed* as well.

4.2.4.2.4 Remote Databases

4.2.4.2.4.1 User/Inbox Shifts published directly to a user are sent here. Upon user connection to the remote publishing server, *user/inbox* is replicated into the local *user/feed*.

4.2.4.2.4.2 User/Messages All messages to a user are sent here. This includes notifications about comments on a shift as well as system notices such as broken shift content and software updates.

4.2.4.2.4.3 System/Messages For efficiency, system messages are stored in their own database and get replicated to client *user/feed*.

4.2.4.2.5 Group Databases

When a group is created by a user, a group database is generated to hold all content shared by the group. When user joins a group, he/she is allowed to replicate the group database into their local *user/feed*. When a user publishes a shift to a group, a copy is sent to the group, so other can replicate that content into their own feeds.

4.2.4.2.5.1 Remote/Master This database holds all public shifts. All of this content is replicated to *remote/shared*. This means each public shift has two copies on the server.

4.2.4.2.5.2 Remote/Shared This database is analogous to the local *user/feed*. This database holds all shared content, whether public, in a group, or sent directly to a user. General queries are run against this database as it is the merge of all of the social activities of the users registered on the publishing server. Again, if a user is running as full peer, queries happen locally for that user not against *remote/shared* [12].

4.2.4.3 Sharding

4.2.4.3.1 Scaling Out

Normally, you start small and grow over time. In the beginning, you might do just fine with one node, but as your data and the number of clients grow, you need to scale out.

For simplicity, we will start fresh and small.

Start node1 and add a database to it. To keep it simple, we will have two shards and no replicas.

```
curl -X PUT "http://xxx.xxx.xxx.xxx:5984/small?n=1&q=2" --user daboss
```

If you look in the directory `data/shards` you will find the two shards.

```
data/
+-- shards/
|   +-- 00000000-7fffffff/
|   |   -- small.1425202577.couch
|   +-- 80000000-ffffffff/
|   |   -- small.1425202577.couch
```

Now, go to the admin panel

```
http://xxx.xxx.xxx.xxx:5986/_utils
```

and look in the database `_dbs`; it is here that the metadata for each database is stored. As the database is called `small`, there is a document called `small` there. Let us look in it. Yes, you can get it with `curl` too:

```
curl -X GET "http://xxx.xxx.xxx.xxx:5986/_dbs/small"

{
  "_id": "small",
  "_rev": "1-5e2d10c29c70d3869fb7a1fd3a827a64",
```

```

"shard_suffix": [
  46,
  49,
  52,
  50,
  53,
  50,
  48,
  50,
  53,
  55,
  55
],
"changelog": [
  [
    "add",
    "00000000-7fffffff",
    "node1@xxx.xxx.xxx.xxx"
  ],
  [
    "add",
    "80000000-ffffffff",
    "node1@xxx.xxx.xxx.xxx"
  ]
],
"by_node": {
  "node1@xxx.xxx.xxx.xxx": [
    "00000000-7fffffff",
    "80000000-ffffffff"
  ]
},
"by_range": {
  "00000000-7fffffff": [
    "node1@xxx.xxx.xxx.xxx"
  ],
  "80000000-ffffffff": [
    "node1@xxx.xxx.xxx.xxx"
  ]
}
}

```

- `_id` is the name of the database.
- `_rev` is the current revision of the metadata.
- `shard_suffix` are the numbers after small and before `.couch`. The number of seconds after UNIX epoch that the database was created. Stored in ASCII.
- `changelog` is self-explaining. Only for admins to read.
- `by_node` are shards which each node have.
- `by_rage` are nodes on which each shard is.

4.2.4.4 Consistency

A *distributed system* is a system that operates robustly over a wide network. A particular feature of network computing is that network links can potentially disappear, and there are plenty of strategies for managing this type of network segmentation. CouchDB differs from others by accepting eventual consistency, as opposed to putting absolute consistency ahead of raw availability like RDBMS or Paxos. What these systems have in common is an awareness that data act differently when many people are accessing it simultaneously. Their approaches differ when it comes to which aspects of *consistency, availability, or partition tolerance* they prioritize.

Engineering distributed systems is tricky. Many of the caveats and “gotchas” you will face over time are not immediately obvious. We do not have all the solutions, and CouchDB is not a panacea; but when you work with CouchDB’s grain rather than against it, the path of least resistance leads you to naturally scalable applications.

Of course, building a distributed system is only the beginning. A website with a database that is available only half the time is next to worthless. Unfortunately, the traditional relational database approach to consistency makes it very easy for application programmers to rely on global state, global clocks, and other high-availability no-nos, without even realizing that they are doing so. Before examining how CouchDB promotes scalability, we will look at the constraints faced by a distributed system. After we have seen the problems that arise when parts of your application cannot rely on being in constant contact with each other, we will see that CouchDB provides an intuitive and useful way for modeling applications around high availability.

4.2.4.4.1 Local Consistency

Before we attempt to understand how CouchDB operates in a cluster, it is important that we understand the inner workings of a single CouchDB node. The CouchDB API is designed to provide a convenient but thin wrapper around the database core. By taking a closer look at the structure of the database core, we will have a better understanding of the API that surrounds it.

4.2.4.4.2 Distributed Consistency

Maintaining consistency within a single database node is relatively easy for most databases. The real problems start to surface when you try to maintain consistency between multiple database servers. If a client makes a write operation on server *A*, how do we make sure that this is consistent with server *B*, or *C*, or *D*? For relational databases, this is a very complex problem with entire books devoted to its solution. You could use multimaster, master/slave, partitioning, sharding, write-through caches, and all sorts of other complex techniques.

4.2.4.4.3 Incremental Replication

CouchDB’s operations take place within the context of a single document. As CouchDB achieves eventual consistency between multiple databases by using incremental replication, you no longer have to worry about your database servers being able to stay in constant communication. Incremental replication is a process where document changes are periodically copied between servers. We are able to build what is known as a *shared nothing* cluster of databases where each node is independent and self-sufficient, leaving no single point of contention across the system.

Need to scale out your CouchDB database cluster? Just throw in another server.

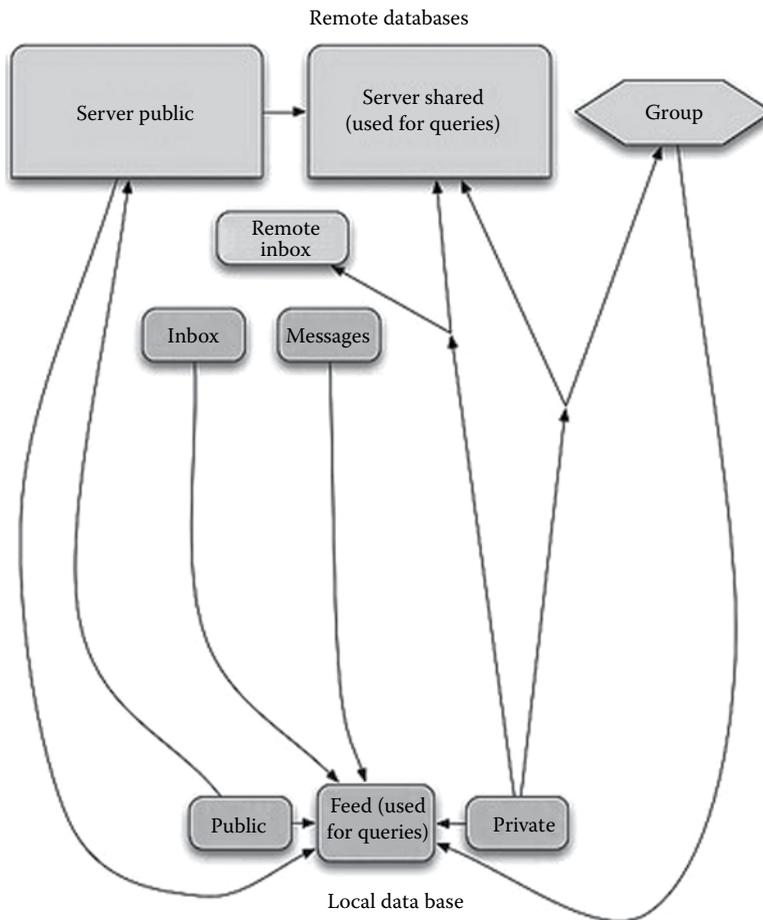


FIGURE 4.10
Incremental replication between CouchDB nodes.

As illustrated in [Figure 4.10](#), with CouchDB's incremental replication, you can synchronize your data between any two databases however you like and whenever you like. After replication, each database is able to work independently.

You could use this feature to synchronize database servers within a cluster or between data centers using a job scheduler such as *cron*, or you could use it to synchronize data with your laptop for offline work as you travel. Each database can be used in the usual fashion, and changes between databases can be synchronized later in both directions.

What happens when you change the same document in two different databases and want to synchronize these with each other? CouchDB's replication system comes with automatic conflict detection *and* resolution. When CouchDB detects that a document has been changed in both databases, it flags this document as being in conflict, much like they would be in a regular version control system.

This is not as troublesome as it might first sound. When two versions of a document conflict during replication, the *winning* version is saved as the most recent version in the document's history. Instead of throwing the *losing* version away, as you might expect, CouchDB saves this as a previous version in the document's history so that you can access

it if you need to. This happens automatically and consistently, so both databases will make exactly the same choice.

It is up to you to handle conflicts in a way that makes sense for your application. You can leave the chosen document versions in place, revert to the older version, or try to merge the two versions and save the result.

4.2.4.5 Replication and Failure Handling

Replication synchronizes two copies of the same database, allowing users to have low-latency access to data no matter where they are. These databases can live on the same server or on two different servers—CouchDB does not make a distinction. If you change one copy of the database, replication will send these changes to the other copy.

Replication is a one-off operation: you send an HTTP request to CouchDB that includes a *source* and a *target* database, and CouchDB will send the changes from the source to the target. That is all. Granted, calling something world class and then only needing one sentence to explain it seem odd. But, part of the reason why CouchDB's replication is so powerful lies in its simplicity.

Let us see what replication looks like:

```
POST /_replicate HTTP/1.1

{"source":"database","target":"http://example.org/database"} -H
"Content-Type: application/json"
```

This call sends all the documents in the local database `database` to the remote database `http://example.org/database`. A database is considered “local” when it is on the same CouchDB instance you send the `POST /_replicate HTTP` request to. All other instances of CouchDB are “remote.”

If you want to send changes from the target to the source database, you just make the same HTTP requests, only with source and target database swapped. That is all.

```
POST /_replicate HTTP/1.1

Content-Type: application/json

{"source":"http://example.org/database","target":"database"}
```

A remote database is identified by the same URL you use to talk to it. CouchDB replication works over HTTP using the same mechanisms that are available to you. This example shows that replication is a *unidirectional* process. Documents are copied from one database to another and not automatically vice versa. If you want *bidirectional* replication, you need to trigger two replications with *source* and *target* swapped [11].

References

1. Ramanathan, S., Goel, S., and Alagumalai, S. 2011. Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable. *2011 International Conference on Recent Trends in Information Systems*. Doi: 10.1109/retis.2011.6146861.
2. Chaganti, P. and Helms, R. 2011. *Amazon SimpleDB: Lite: What is SimpleDB, How Does it Compare to Relational Databases, and How to Get Started?* Birmingham: Packt Publishing.
3. Chaganti, P. and Helms, R. 2010. *Amazon SimpleDB Developer Guide Scale Your Application's Database on the Cloud Using Amazon SimpleDB*. Birmingham: Packt Publishing
4. Fowler, A. n.d. *NoSQL for Dummies*. US: Wiley.
5. Yadava, H. 2007. *The Berkeley DB Book*. Berkeley, CA: Apress.
6. Vrščaj, J. and Demšar, J. 2010. *Razvoj aplikacij na platformi Google App Engine=: Diplomsko delo*. Ljubljana: J. Vrščaj.
7. Stawart, D. T. 2012. *NoSQL: Computing, Database Management System, Relational Database Management System*. Beau Bassin: Dicho.
8. Chodorow, K. n.d. *MongoDB: The Definitive Guide*. US: O'Reilly Media.
9. Boeker, M. 2010. *Mongodb*. Germany: Entwickler Press.
10. Introduction to MongoDB. 2010. *The Definitive Guide to MongoDB*, 3–17. Doi: 10.1007/978-1-4302-3052-6_1. New York City: Apress.
11. *CouchDB the Definitive Guide*. 2009. Sebastopol, CA: O'Reilly Media.
12. Couchdb. 2012. S.l.: Book On Demand.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

5

NoSQL and Cloud Paradigm: Characteristics and Classification, Data Storage Technology, and Algorithms

Ajanta Das, Anindita Desarkar, and Mridul Paul

CONTENTS

5.1	Introduction	99
5.2	Related Work	100
5.3	Cloud Execution Models.....	100
5.3.1	Task Scheduling in Cloud.....	101
5.3.1.1	List Scheduling.....	101
5.3.1.2	Directed Acyclic Graph Scheduling.....	102
5.3.2	Data Storage Technology and Algorithms in Cloud.....	103
5.3.2.1	Programming Paradigm for Cloud Applications.....	103
5.3.2.2	Cloud Storage Architecture	106
5.3.3	Distributed Data Storage Algorithms and Efficient Data Storage Strategies	109
5.3.3.1	Distributed Data Storage Algorithms	109
5.3.3.2	Efficient Data Storage Strategies in Cloud	110
5.4	Performance Management in NoSQL.....	114
5.4.1	Performance Measurement of NoSQL Database in Cloud Platform.....	114
	Bibliography	116

5.1 Introduction

Relational databases are based on traditional data storage concepts that come with limitations. These limitations include issues in handling globally distributed data which are semistructured in nature, available in real time, and voluminous. Also, these traditional databases have limited processing power and storage capacity. The main features of NoSQL include higher scalability, adequately optimized for distributed computing, elastic schemas, etc. NoSQL databases solve these challenging issues with various data model, storage structure, and access patterns.

NoSQL supports nonrelational data model and is able to handle a wide variety of data structures. Information can be stored without schema. This feature is highly compatible with cloud computing technology where there is a requirement to support heterogeneous kind of data. In cloud computing world, data are stored across the globe in various servers, and their size is growing rapidly. In this scenario, maintaining the relationship across the data set is a big challenge but is mandatory for all relational databases. Here, also NoSQL

can be the one-stop solution as it does not require building the relationship between their records. It automatically increases the performance as it reduces the query time, which is done by joining tables containing different data structures. NoSQL is also highly distributable in nature, which can store and process information present in various servers, and a cluster of servers can be used to hold a single database. Due to the above-mentioned characteristics, NoSQL database is chosen for storage and retrieval of data in cloud.

The structure of the chapter is organized as follows. Section 5.2 presents related work. Section 5.3 presents various execution models in cloud: Cloud Storage Architecture and Distributed Data Storage Algorithms, and Efficient Data Storage Strategies. Performance Measurement of NoSQL Database in Cloud platform is presented in Section 5.3, and Section 5.4 concludes the chapter.

5.2 Related Work

This chapter studies a few current research based on NoSQL and cloud paradigm. Cloud execution models are significantly driven by underlying task-scheduling algorithms. Singh and Kumar (2014) did an elaborated comparison of different scheduling algorithms based on the fitment, viability, and adaptability in context of cloud computing. A hybrid algorithm is proposed that can adapt to the existing cloud platforms for improving Quality of Service (QoS). However, in another research paper by Salot (2013), both static and dynamic algorithms are judged based on certain key parameters such as make span, processing times, throughput, and QoS. Issues are highlighted for algorithms that are considered to be high on availability and reliability. Some algorithms such as Directed Acyclic Graph (DAG) scheduling (Salot, 2013) are further analyzed using simulated data for its fitment in tackling scheduling problems. Efficient programming models, such as Message-Passing Interface (MPI) and Map Reduce, have been critiqued by Geist (1994) and Barker (2015). Both these researches provide pragmatic approach using such models and tie these models with scheduling underlying computing resources. The next section discusses cloud execution models and applications.

5.3 Cloud Execution Models

At the outset, it is important to brief on basic tenets of cloud before looking at the execution models in cloud. Cloud is grounded on the tenets of *virtualization*, *distributed computing*, and *service-oriented computing*. Virtualization provides an abstract environment that allows users to run their applications (Paul and Das, 2015). This environment can include hardware resources and software resources based on the levels of abstraction. Distributed computing provides immense capabilities to integrate independent computing resources to appear as a coherent system (both at hardware and at OS layer) to the users. The concept of services in cloud is derived from Service-Oriented Computing where a set of design principles provide building blocks for system development and structure for integrating components into a coherent and decentralized system.

5.3.1 Task Scheduling in Cloud

Execution models in cloud are essentially centered on task execution. Task in Cloud activity is expected to be done using underlying resources. The key to this execution is to use optimal resources with minimum execution time (Singh et al., 2014). Since there will be several tasks that need to be handled by cloud resources at a time, it becomes critical to schedule tasks efficiently. Task scheduling is grouped under two types: *static* and *dynamic* scheduling. Static scheduling refers to scheduling of tasks where information regarding complete sequence of tasks, execution times, and resource mapping is known before execution. Dynamic scheduling, on the other hand, requires system current states of environment and computing machines to make scheduling decision. Virtualization techniques are deployed in cloud to map computing resources to run scheduling algorithms. Virtual Machines (VMs) provide abstraction of computing and storage resources made available to scheduling algorithms to execute tasks. These algorithms are usually managed by a broker that coordinates with virtualization layer to manage VMs. Several common approaches of task scheduling in cloud are explained in the following.

5.3.1.1 List Scheduling

The fundamental tenet of list scheduling is organizing a task list through a set of policies involving priorities. These priorities are governed by the algorithm approach that manages these lists. The execution of tasks is in the order of the highest-priority task, assigning a resource (VM). Some lists scheduling algorithms are summarized below in this section.

5.3.1.1.1 First-Come-First-Serve Algorithm

In this algorithm (Salot, 2013), the list is treated as task queue. The tasks received by the broker are queued in a linear list. As the tasks arrive, they are added to the end of the queue. The broker selects first task and scans through available pool of VMs that can execute this task. Algorithm does not validate whether VMs selected are the most efficient to perform the execution. Though this algorithm is simple, it is not efficient in terms of utilization of VMs. Scenario of this algorithm is depicted in Figure 5.1.

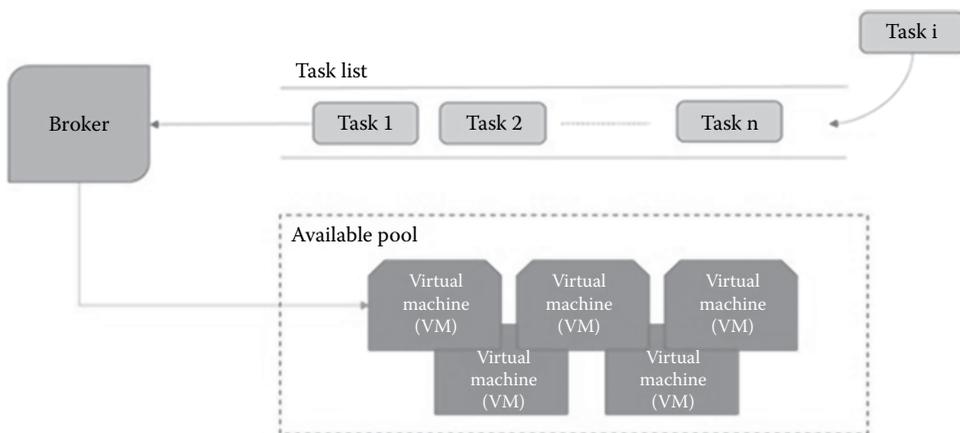
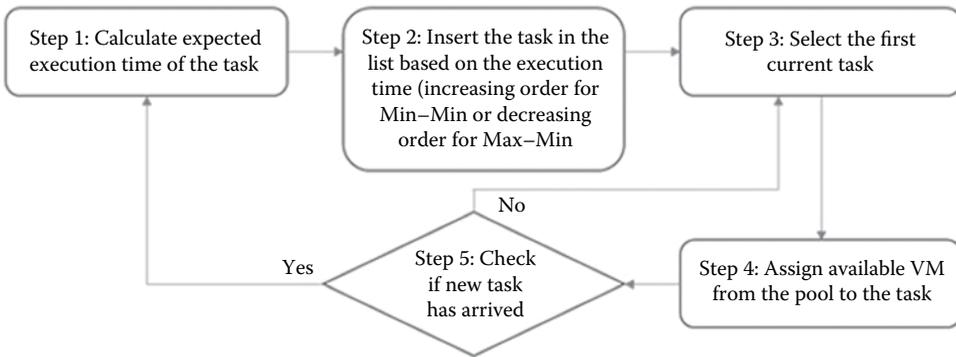


FIGURE 5.1
First-come-first-serve algorithm.

**FIGURE 5.2**

Min-Min and Max-Min algorithm steps.

5.3.1.1.2 Round-Robin Algorithm

Round-robin algorithm manages task execution in a *First-In-First-Out (FIFO)* manner; however, each task is given a limited CPU time slice in a given VM (Salot, 2013). So, in the event of a task that is not completed during the time slice, the CPU is released for the next task waiting for execution in the queue. The preempted task is then placed at the back of the ready list.

5.3.1.1.3 Min-Min and Max-Min Algorithm

Min-Min algorithm calculates execution time for tasks and creates a list of tasks in an increasing order of their execution time. Thus, small tasks are chosen to be executed first. The broker allocates available VMs for tasks in that order. As more tasks arrive, the algorithm inserts the task in existing sequence based on the calculated execution time. However, the drawback is that the larger tasks which may be critical are executed at last in the sequence. Max-Min algorithm, however, does the reverse. This algorithm executes large tasks first, which in turn causes further delay in execution of smaller tasks. Steps of the algorithm are presented in [Figure 5.2](#).

5.3.1.2 Directed Acyclic Graph Scheduling

One of the basic assumptions in list scheduling is that there are no interdependence tasks. Each task that arrives in the queue is independent and exclusive to any other task in the queue. DAG scheduling can handle such interdependence. DAG is a generic model for a parallel program that consists of a set of processes (nodes) and dependencies (edges). In general, a DAG is defined by the tuple $G = (V, E)$, V ($|V| = v$) is a set of nodes representing the tasks, and E ($|E| = e$) is a set of directed edges representing communication messages (Kalisch and Bühlmann, 2007). A simple use case is considered here for DAG scheduling in cloud. [Figure 5.3](#) shows a dependence DAG for a set of tasks and an available pool of VMs. Each node (i.e., task) in the DAG carries weight w_i that represents execution time for a node i . Each VM has its own memory and also connected to a shared memory for storing common messages among the tasks.

The time required to execute a task in one VM is T_1 , and the time required to execute all the tasks on critical path is T_4 . The scheduling algorithm can be mapped in terms of space (number of VMs) and time. [Figure 5.4](#) depicts the scenarios that can be derived based on

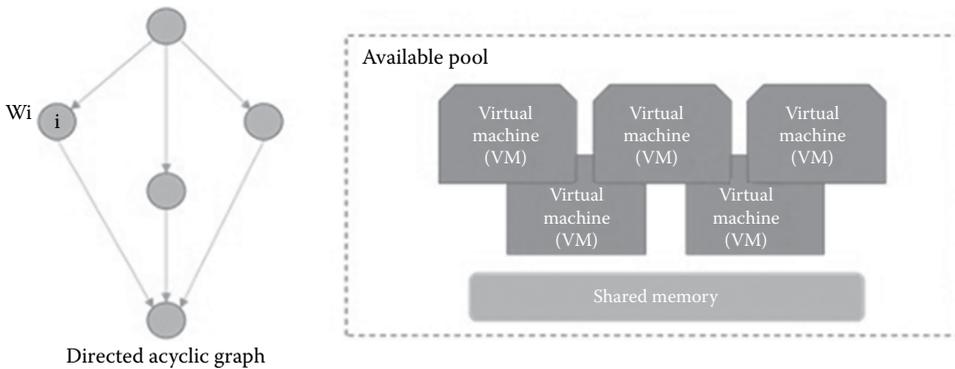


FIGURE 5.3 DAG and available pool of VMs.

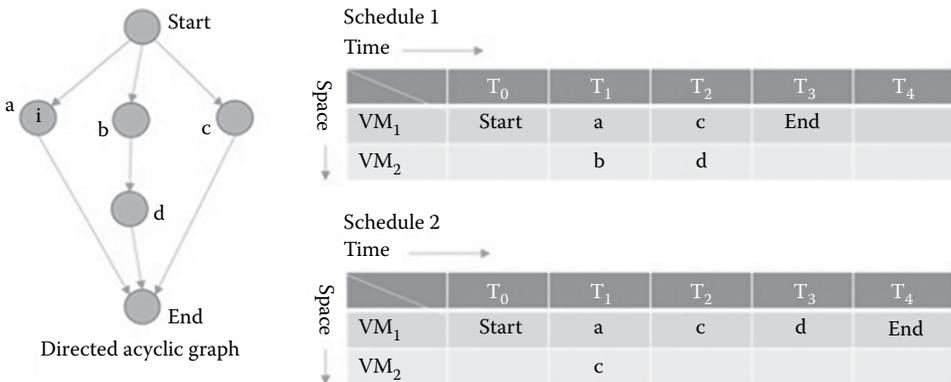


FIGURE 5.4 DAG scheduling scenarios using two VMs.

the space-time mapping considering two VMs allocated for four DAG tasks. Assuming that each task takes timeslot of T1 (where T1 = T2 = T3 = ...), Schedule 1 represents the optimal timelines for completing all tasks with dependencies intact (Task b needs to complete before Task c). Schedule 2 depicts sequential pick of network (Route: Start → a → End, then Route: Start → b → d → End and at least route: Start → c → End). This schedule may not be optimal compared to Schedule 1, but has that constraint that needs to be complied with. [Figure 5.5](#) represents steps for simple DAG scheduling algorithm for allocating VMs in cloud.

5.3.2 Data Storage Technology and Algorithms in Cloud

This section explains data storage technology and algorithms in cloud. Detailed discussion of programming paradigm is essential for learners in cloud computing. Hence, before explaining the details of cloud databases, this section describes the different types of programming paradigms.

5.3.2.1 Programming Paradigm for Cloud Applications

In reality, cloud computing is used extensively in large data processing applications. Government institutions and large enterprises are leveraging cloud infrastructure to

```

Step 1:   cycle c = 0;
Step 2:   ready-list = {START};
Step 3:   inflight-list = { } ;
Step 4:   while (|ready-list|+|inflight-list| > 0) {
Step 5:       for each node n in ready-list in priority order { //schedule new tasks
Step 6:           if (a processor is free at this cycle) {
Step 7:               remove n from ready-list and add to inflight-list;
Step 8:               add node to schedule at time cycle;
Step 9:           } else break;
Step 10:          c = c + 1; //increment time
Step 11:          for each node n in inflight-list { //determine ready tasks
Step 12:              if (n finishes at time cycle) {
Step 13:                  remove n from inflight-list;
Step 14:                  add every ready successor of n in DAG to ready-list

```

FIGURE 5.5

DAG scheduling algorithm. (Adapted from Kwok, Y. K. and Ahmad, I. 1999. *ACM Computing Surveys (CSUR)*, 31(4), 406–471.)

process data in efficiently and at faster rate. Major emphasis is on using parallel programming models to derive extreme capabilities of computing and storage hardware. Below are some of the interesting programming models for cloud applications.

5.3.2.1.1 Parallel Virtual Machine

Parallel Virtual Machine (PVM) is a programming model that provides framework for connecting computing devices in a network. The design provides abstraction of several heterogeneous machines to form a single computing entity. The underlying technique for PVM is message-passing model that exploits distributed computing across heterogeneous computing devices, including Massively Parallel Processing (MPP). This model handles message routing, data conversion, and task scheduling among the computing devices. The PVM library (Geist, 1994) is portable and available as open source in the Internet. The user can construct one or more sequential programs in different programming languages such as C, C++, or Fortran 77, containing calls to PVM library. Each program represents a task for a given application. These programs are compiled in a host pool of computing machines, and the compiled objects are kept in a location that is accessible from machines in the host pool. In order to execute an application, the tasks are triggered from any machine within the host pool. Machines are usually using *Single Program Multiple Data (SPMD)* type of computing model. This model then starts other PVM tasks that result in a collection of active tasks. These tasks are computed on different machines and can exchange messages with each other to solve the problem. PVM programming model is displayed in [Figure 5.6](#). This model can be used at *Infrastructure as a Service (IaaS)* layer to utilize resources and perform data-intensive computations.

5.3.2.1.2 Message-Passing Interface

MPI provides interface for communicating messages among a cluster of computing machines. The MPI interface specifies the functionality of high-level communication routines. Functions give access to a low-level implementation that takes care of sockets,

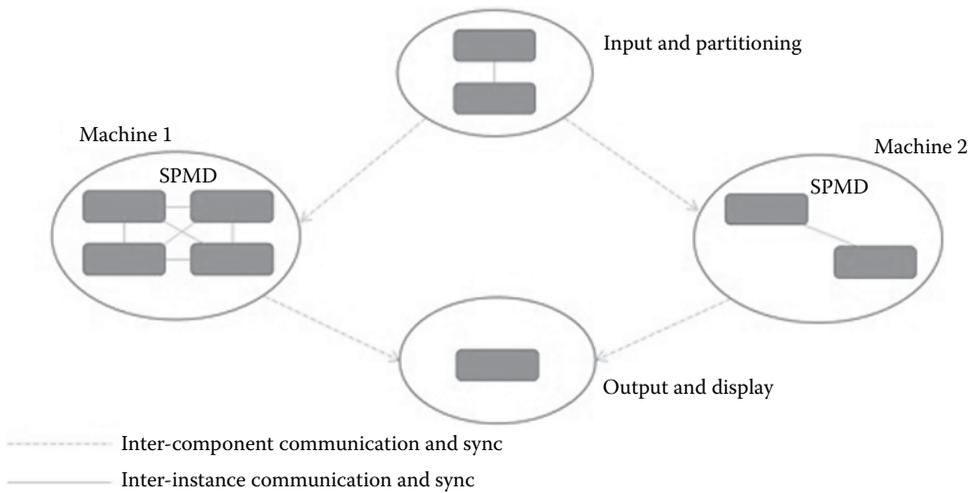


FIGURE 5.6
PVM programming model.

buffering, data copying, message routing, etc. (Barker, 2015). MPI applications are fairly portable and an efficient way of utilizing computing infrastructure. Some of the open MPI implementations are MPICH and OpenMPI. An illustration of a simple algorithm that invokes MPI interface to perform parallel computations is given in Figure 5.7. The figure represents an algorithm using MPI interface calls.

5.3.2.1.3 Map Reduce

This programming model is currently adopted for processing large sets of data. The fundamental tenets of this model are map and reduce functions. The function of Map is to generate a set of intermediate key and value pairs, while Reduce function merges all intermediate values with same key. The model provides an abstract view of flow of data and control, and the implementation of all data flow steps such as data partitioning, mapping,

- Step 1:** Include the implementation-specific header file
#include inserts basic definitions and types
- Step 2:** Initialize communications –
MPI_Init initializes the MPI environment
MPI_Comm_size returns the number of processes
MPI_Comm_rank returns this process's number (rank)
- Step 3:** Communicate to share data between processes –
MPI_Send sends a message
MPI_Recv receives a message
- Step 4:** Exit from the message-passing system –
MPI_Finalize

FIGURE 5.7
An algorithm using MPI interface calls.

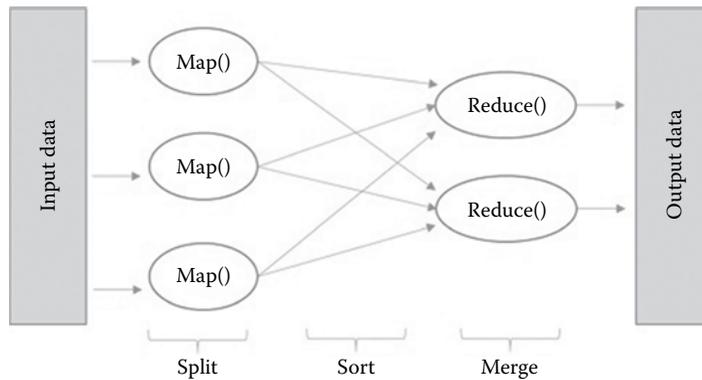


FIGURE 5.8
Map Reduce programming model.

synchronization, communication, and scheduling is made transparent to the users. User applications can use these two functions to manipulate the data flow. Data-intensive programs that are based on this model is represented in [Figure 5.8](#), can be executed *PaaS*. Hadoop, an open implementation by Apache, is based on Map Reduce model that has been used by many companies such as AOL, Amazon, Facebook, Yahoo, and New York Times for running their business application (Rao and Reddy, 2012).

5.3.2.2 Cloud Storage Architecture

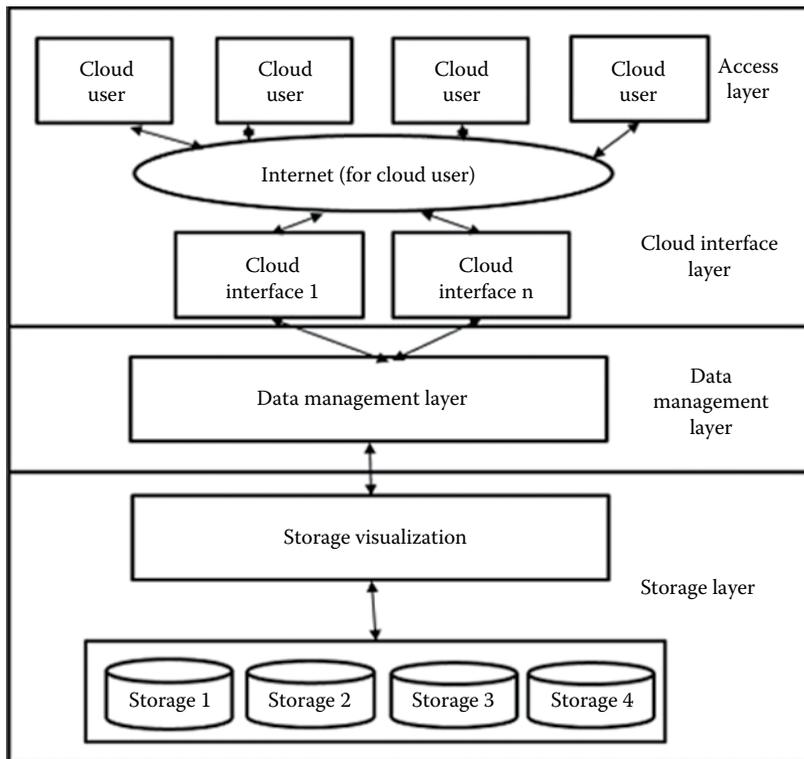
Cloud storage is more about applications than just storage. In the world of cloud computing, cloud storage deals with the process to control and manage all kind of data and information with hardware, service (software), and platform resources. Therefore, in order to provide efficient cloud service to user, identifying and organizing the cloud storage model are very important task (Jeong and Park, 2012).

Reliable data center services are responsible for the transmission of most of the cloud computing infrastructure, and server virtualization technology is the basis for creation of various levels. Network infrastructure services can be provided to any place for using these services. Cloud is the single access point for fulfilling the computing need of all users. Cloud storage consists of application software along with the storage device where the application software can leverage the storage service to keep the changes. Cloud storage does not consist only of the hardware like the traditional storage devices, but it also is the combination of network and storage equipment, servers, clients, applications, and interface for common access, accessing network along with other system components.

The general architecture of cloud storage is presented in [Figure 5.9](#), and it mainly includes four layers mentioned in the following.

5.3.2.2.1 Access Layer

Access layer is the primary interaction point for applications and cloud storage users which is accessible to the authorized user through a standard application interface from anywhere across the globe. This layer is also called as data service layer. There are various cloud storage operating units which facilitate various access types to the cloud storage systems, for example, web access.

**FIGURE 5.9**

Generalized architecture of cloud storage. (Adapted from Elzeiny, A., Elfetouh, A. A., and Riad, A. 2013. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 2(4), 342–349.)

5.3.2.2.2 Cloud Interface Layer

Cloud storage providers provide a software layer to create the connection between cloud storage service and cloud user via Internet. User authorization and authentication are executed in this layer. This is also known as application interface layer or data service layer. This layer can be flexibly expanded and directly user-oriented. There are various application interfaces created to provide various services like data storage, video surveillance, network drivers, and multiuser sharing depending on various types of businesses and user requirements.

5.3.2.2.3 Data Management Layer

Managing a specific cloud client is the principal job for its software layer. This layer is responsible for producing an integrated view of public administration required for the upper cloud storage system. This integrated public data management interface ensures the correct connectivity between storage layer and application interface layer. Here, clustered system, distributed file system, and grid computing technology are used to provide stronger storage devices, which is based on the implementation of coordinating various storage devices in cloud storage system.

Data management includes the following activities:

- Storing data
- Distribution of content across storing locality

- Partitioning of data
- Synchronizing
- Consistency maintenance
- Reproduction
- Controlling data flow over network
- Backup
- Recovery of data
- Large number of user handling
- Metadata and catalog maintenance

5.3.2.2.4 Storage Layer

The most preliminary part of cloud data storage is Storage layer. Virtualization and Basic storage are the two main sections of this layer (Raj, 2014).

5.3.2.2.4.1 Basic Storage Basic storage consists of database servers and heterogeneous storage devices across various locations, where these servers or storage devices are connected through wide area network, Internet.

5.3.2.2.4.2 Storage Virtualization Storage virtualization is a technique for storing various types of storage devices and managing as an integrated single unit so that a single view of storage appears. It maps from heterogeneous storage devices, which are also distributed in nature, to a specific storage location, continuous in nature, and also responsible for creating a shared dynamic platform. Two commonly used storage virtualization techniques are storage tiering and thin provisioning.

- *Thin provisioning*: In the current world, the biggest challenge is managing a huge amount of data and allocating appropriate storage space according to the requirement so that the challenges like over-provisioning and wastage of storage capacity are handled appropriately. Thin provisioning or virtual provisioning comes into picture to address this challenge. It basically refers the provisioning as per the requirement. Here, an application is allocated storage from a common pool based on the need. With this thin provisioning, usually logical storage is allocated by storage administrator to an application, but in reality, the system releases the actual physical capacity only at the time of requirement. This feature provides an optimal utilization of storage by reducing the amount of allocated but unused physical storage.
- *Storage tiering*: Current techniques should be in place that enables to store right data in right location and make them accessible in the correct time by the organizations. These things can be achievable by applying the technique of storage tiering. Hierarchy of storage types can be established where data can be stored in the hierarchy based on their performance, availability, and recovery requirements. As an example, drives with high performance can be configured as tier1 storage for keeping frequently accessed data, which will improve performance. On the other hand, low-cost drives can be configured as tier2 for residing data that are not accessed frequently. Here, tier1 space is freed up, which increases the capacity of high-performance drives and reduces the cost of storage. These data movement happens based on few predefined properties, like file type, access frequency, and performance.

The main *benefits of storage virtualization* include the following:

- To provide cost-effective ways for eliminating single point of failure in the storage area network.
- Improvement of service quality by applying proper management and process improvement technique in real time.
- Disaster recovery and data archival can be achieved by providing cost-effective solutions.
- Utilization, scalability, and flexibility improvement across the storage area network.

5.3.3 Distributed Data Storage Algorithms and Efficient Data Storage Strategies

Data storage in cloud platform enables user to store and retrieve data in a remote and distributed fashion without having the burden of hardware and software in their local machines along with the availability of on-demand, high-quality cloud applications. As the whole process is online, users can access it from any location via Internet. There are a few top advantages of cloud data storage, which include the following:

- I. *Flexibility*: Users are allowed to drag and drop files between the cloud storage and their local storage.
- II. *Accessibility*: The stored files in the cloud can be accessed from anywhere across the globe via Internet.
- III. *Disaster Recovery*: As the files are stored in the remote location, they are safe in case of any disaster that causes damage to local files that consists of critical business data.
- IV. *Cost savings*: As storage can be paid based on usage, expenditure can be reduced while data volume is low.

5.3.3.1 Distributed Data Storage Algorithms

Cloud-based storage has few inherent vulnerabilities. The biggest challenges in the cloud storage include data security, reliability, and data access efficiency. The challenges along with their solutions are presented below.

5.3.3.1.1 Data Security

Data security is a common threat in any technology domain, but it is more vulnerable to the cloud infrastructure. Various security issues may arise in the areas like external data storage, dependency on the “public” Internet and absence of control, resource sharing and integration with internal security. Encryption technique is the most common and effective way to protect sensitive data from the past. So, security is much assured if stored data or the data that are sent are appropriately encrypted. In that case, encryption algorithms should be reasonably strong. Few encryption schemes such as advanced encryption standard (AES), RSA, etc. are widely used in the market.

5.3.3.1.2 Reliability

According to NIST, reliability is broadly a function of reliability of four individual components: the hardware and software facilities offered by providers, the provider’s personnel, and connectivity to the subscribed services and the consumer’s personnel (Walker, 2012).

The preliminary concern is movement of data from cloud storage provider to the cloud user in a reliable manner. Reliable data movement ensures successful transfer of data from source to target without any malicious attacks in the middle way.

5.3.3.1.3 Data Access Efficiency in Cloud

The biggest advantage of cloud storage lies in its flexibility, which allows users to access data at any time. Here, in the cloud system, user requirements are analyzed automatically by storage management system. It also finds and transforms data which helps the users to a great extent. But, cloud management system proposes high demands for itself. As an example, an incident happened in July 2008 where a service had failed in Simple Storage Service for 8 hours. It caused a great loss for online companies who are dependent on S3. The main reason for the failure was that the S3 system was unable to route the user's request to the proper physical storage server. So, optimization in cloud storage is highly required to ensure the data storage and efficient access. The Huffman coding is an algorithm that can be used to better data access performance in cloud environment.

- Improvement of data access performance in cloud by the Huffman coding:

The Huffman encoding is used for the lossless compression of files which is based on the occurrence frequency of a symbol present in the file which is getting compressed. Statistical coding is the basis of the Huffman algorithm, which means that the probability of a symbol has a direct impact on the length of its representation. The summary is assigning short code words to the input blocks with high probabilities and long code words to the blocks of low probabilities.

The algorithm is described clearly with the help of an example in [Figure 5.10](#).

[Table 5.1](#) describes a few problems that arise due to data storage in cloud platform and their respective solutions and how those can be removed or minimized (Govinda and Kumar, 2012; Jadhav and Nandwalkar, 2015).

5.3.3.2 Efficient Data Storage Strategies in Cloud

There are many storage efficient technologies out of which thin provisioning, automated tiering, and data reduction techniques are the most popular ones (Jeong and Park, 2012).

Step 1: Suppose a file has the data XXXXXYYZZ which should be compressed.

Step 2: Find the frequency of every distinct character in the file. Here, the frequency of 'X' is 6, and the frequency of 'Y' is 4 and the frequency of 'Z' is 2.

Step 3: Find the total number of bits required to store this file. Here, if each character is represented using a fixed length code of two bits, and the total number of bits required to store this complete data will be:
 $(2 * 6) + (2 * 4) + (2 * 2) = 24$.

Step 4: Assign the code based on the frequency of the characters. Here, 'X' has the highest frequency, represented by code 0 (1 bit); 'Y' has the second highest frequency, represented by code 10 (2 bits); and 'Z' has the lowest frequency, represented by code 11 (2 bits).

Step 5: Find the modified size of the file. Here, it will be $(1 * 6) + (2 * 4) + (2 * 2) = 18$. So, file is compressed.

FIGURE 5.10

Steps for the Huffman coding algorithm.

TABLE 5.1

Various Issues of Data Storage in Cloud Platform and Solution Overview

Challenges in Data Storage in Cloud Platform	Mitigation Techniques	Brief Description of Implementation Technique
Poor data access performance over cloud	Optimize cloud storage by compressing data during storage so that data transfer time and storage reduced subsequently, which improves the data access performance over cloud—by using compression algorithm.	Huffman coding—entropy encoding algorithm used for lossless data compression. LZ77—Achieve compression by replacing repeated occurrences of data with reference to a single copy of that data existing earlier in the input (uncompressed) data stream. LZ78—Achieve compression by replacing repeated occurrences of data with reference to a dictionary that is built based on the input data stream.
Data security issue: privacy preserving issue between third party auditor and the data in the cloud	By implementing encryption algorithm.	Prominent encryption techniques that use AES encryption algorithm, based on several substitutions, permutations, and linear transformations, each executed on data blocks of 16 bytes.
Data security	Encrypt data to protect from unauthorized user access. Security mechanism in distributed data storage using proxy re-encryption.	Apply RSA—data encryption algorithm to secure data. Using this concept re-encryption, the data get more secure and access permission that who will access the data is decided by only the owner of data.
Data management: uneven load distribution in cloud	Implementing appropriate load-balancing process—load balancing is a process by which the total load of the collective system is assigned among the individual nodes to make the optimum utilization of the resources.	Redistribution of load in cloud using improved distributed load-balancing algorithm with security.

5.3.3.2.1 Thin Provisioning

Thin provisioning is an efficient optimization technique by which optimal or near-optimal utilization is achievable to the available space of storage area networks. In this technique, disk storage space is allocated among users in such a way that each user receives the minimum space at any given time. In a traditional storage provisional model, which is also known as “flat provisioning,” storage space is allocated beyond current requirement considering the future need. So, the ultimate outcome is low storage utilization rate. Other related benefits are reduction in consuming of electrical energy, lesser requirement of hardware space, and reduction in heat generation compared with traditional networked storage systems. [Figure 5.11](#) depicts the differences between traditional and thin provisioning.

5.3.3.2.2 Automated Tiering

Automatic tiering is the real-time intelligent mechanism which continuously places data on the appropriate storage class based on the data access frequency. As the correct future prediction is not always possible, auto tiering is extremely useful, which moves data with higher accessing frequency to the faster tier and rarely accessed data to the slowest tier automatically. In Gartner’s report published on April 2013, the effectiveness of automatic tiering is mentioned clearly. According to him, “The access patterns, value and usage

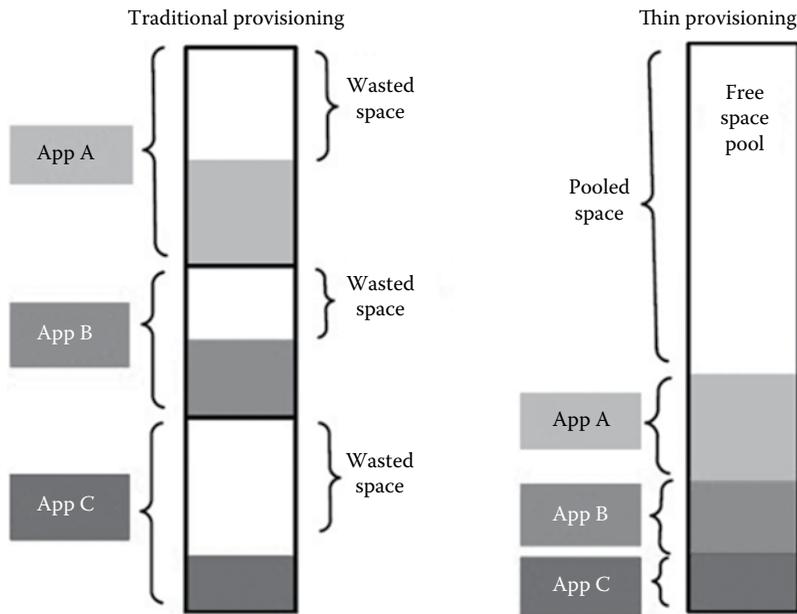


FIGURE 5.11

Traditional provisioning versus thin provisioning. (Adapted from Thin Provisioning by NetApp Community, last edited on 2016. Available at <http://community.netapp.com/t5/Technology/Space-Reclamation-If-Being-Thin-is-Attractive-Then-Thin-Provisioning-Makes-for/ba-p/82836>.)

characteristics of data stored within storage arrays varies widely, and is dependent on business cycles and organizational work processes. Because of this large variability, data stored within storage devices cannot be economically and efficiently stored on the same storage type, tier, format or media.”

The advantages of automatic tiering are listed in the following:

- Provide automatic allocation of fastest tier of storage to the most demanding workload
- Application performance improvement as making contribution to total usable storage capacity

Figure 5.12 describes a tiering model based on application’s performance and capacity. This model consists of archive tier, capacity tier, and performance tier.

From an input/output perspective, the preliminary goal of automated storage tiering is to shift as much random input/output as possible to high-performance media (Flash) for minimization of input/output burden on HDDs, which automatically leads to reduction of average latency.

The distinction between random input/output and sequential input/output is important because Flash has a relatively little price/performance advantage over HDDs for sequential reads and writes because HDDs are good at sequential input/output.

5.3.3.2.3 Data Reduction Techniques

Data reduction is simply deletion of company’s unwanted data so that it will be managed in a more effective way. According to Christine Taylor, an analyst at Taneja Group in

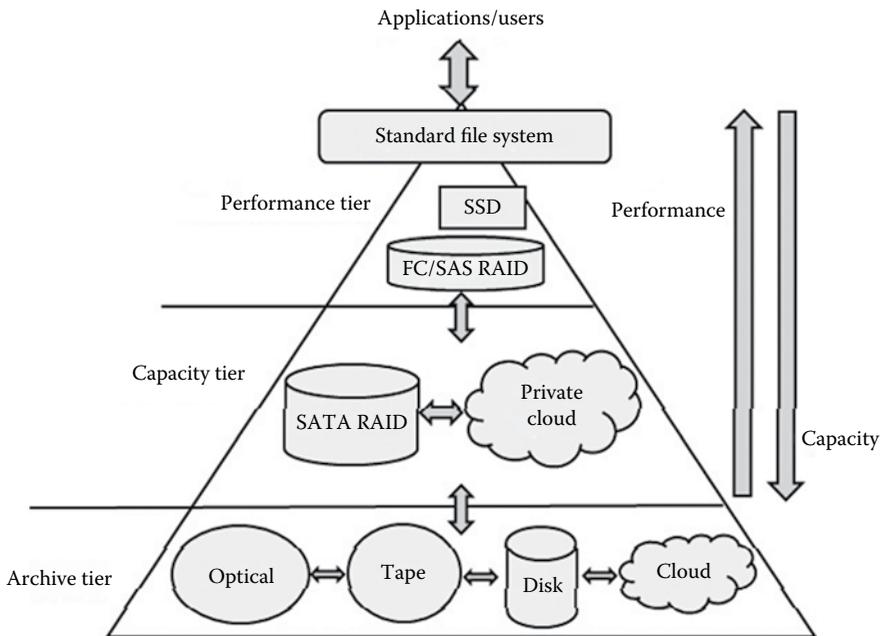


FIGURE 5.12

Automated tiering. (Adapted from Automated Tiering by euroson—Archival and Storage Solutions. Available at <http://www.euroson.com/products/point-software/point-storage-manager/>.)

Hopkinton, “The real savings in energy efficiency comes in managing the data,” so automated data retention strategies should be in place to safely eliminate or copy data to tape at the end of their preset retention periods. Data classification can play an effective role in wiping out unnecessary data.

We can primarily concentrate on the following basic steps for effective data management and reduction:

- *Gather clear understanding on capacity:* Data storage environment needs to be thoroughly profiled so that appropriate places can be identified to increase the efficiencies. This will also help later during the tiering process.
- *Remove unwanted files:* Several duplicate files or irrelevant files stored across the server should be identified and deleted in a regular basis for achieving better performance.
- *Appropriated data tiering:* Organization’s data should be tiered in a proper way.
- *Watch the trends:* Adequate attention should be paid to the utility of the resources along with the storage environments. Proper techniques can be applied accordingly to cope with the changes.

5.3.3.2.4 Data Deduplication Techniques

Deduplication is the technology that shrinks the data footprint by removing redundant data at the file or subfile level. This is the most common technology with backups and archives, but it increases acceptance with primary storage. There are many providers

offering this aspect, but NetApp is the top one with whom more than 8000 customers have already licensed for its free deduplication technology.

5.3.3.2.5 Data Compression Techniques

Data compression reduces the size of data based on the algorithms. It can be done separately or along with data deduplication. Leading storage vendors are already providing a few compression techniques such as Storwize, Inc. that promotes compression of primary storage, and EMC Corp. and Ocarina Networks, Inc. that offer a combination of data deduplication and compression for primary storage.

5.4 Performance Management in NoSQL

Cloud data management system needs to have the characteristics like scalability and high performance, elasticity, fault tolerance, availability, ability to run on commodity heterogeneous servers, etc., which cannot be achieved by the traditional RDBMS. NoSQL is a new kind of data store which is able to fill the gaps and acquires the features of flexible schema, ability to scale horizontally over many commodity servers, and high availability along with many other characteristics.

5.4.1 Performance Measurement of NoSQL Database in Cloud Platform

The definition of performance in cloud computing can be presented based on the definition by ISO 25010:

The performance of a Cloud Computing system is determined by analysis of the characteristics involved in performing an efficient and reliable service that meets requirements under stated conditions and within the maximum limits of the system parameters.

The three concepts that are most important to carry out the performance measurement in cloud computing are consumers, developers, and maintainers. Except that, few terminologies are described below which are the prime factors for measuring performance (Bautista et al., 2012):

- *Performance efficiency*: The amount of resources used under specific circumstances, which includes both the software and the hardware products.
- *Capacity*: Defines the degree at which requirement of a product is met.
- *Resource utilization*: Describes the type and quantity of resources for any particular product or system.
- *Time behavior*: It includes the reply, processing time, and throughput rate of a product to meet the requirements.
- *Reliability*: It measures whether a system can perform specified functions under particular conditions for a certain period of time.
- *Maturity*: It measures whether the reliability requirements are met under normal operation or not.

- *Availability*: The degree at which a system is functional and accessible at the time of requirement.
- *Fault tolerance*: It measures whether the system is operational if any software or hardware failure occurs.
- *Recoverability*: The degree at which a system can recover data directly at the time of failure or interruption and can be restored to the expected condition.

Figure 5.13 presents the relationships between the performances and subconcepts proposed by ISO 25010.

Performance efficiency and reliability are the two main determining factors of system performance. When the system gets the service request, there may be three possible outcomes: the service is correctly performed, performed incorrectly, or cannot be performed.

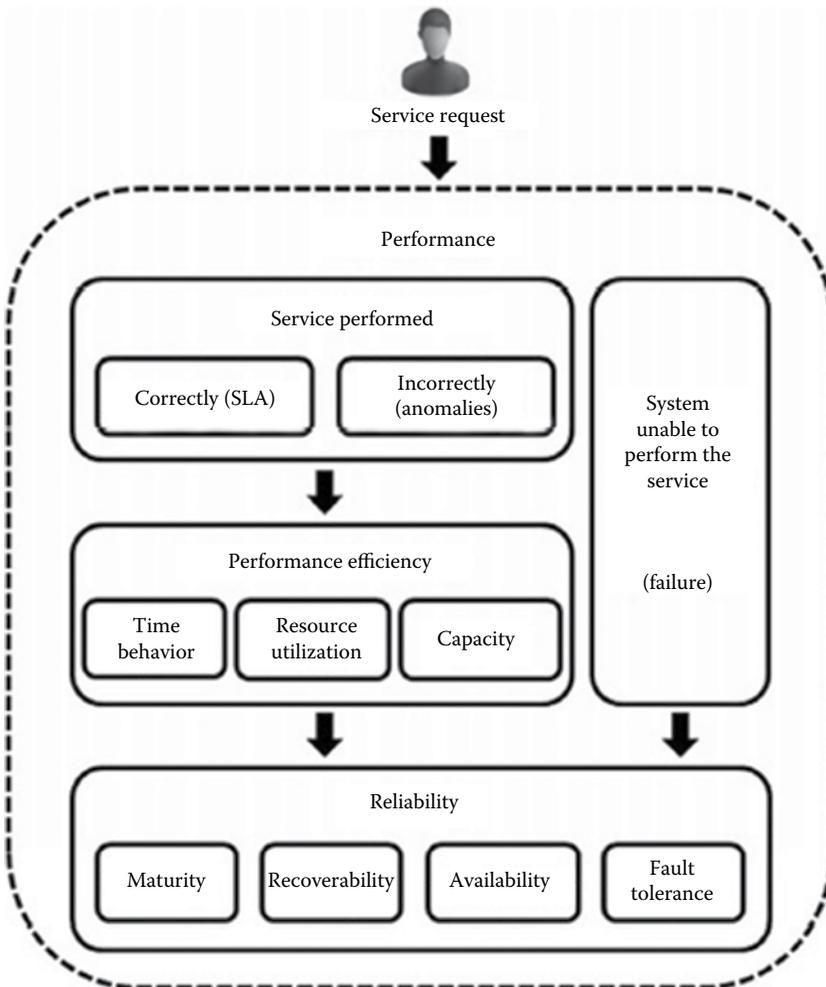


FIGURE 5.13 Performance management in cloud. (Adapted from Bautista, L., Abran, A., and April, A. 2012. *Journal of Software Engineering and Applications*, 5(2), 69–75.)

TABLE 5.2

Performance Parameters in Cloud and Respective NoSQL Feature

Parameters—Performance Measurement in Cloud	NoSQL Feature—Helps to Achieve Better Performance	NoSQL Feature Description—How It Achieves Better Performance
Performance efficiency Capacity Resource utilization	Scalability	NoSQL databases were built for the distributed, scale-out technologies. Distributed set of nodes—known as cluster—is used here, which provides highly elastic scaling capability to the user by which they can add nodes on the fly if required. This unique feature enables NoSQL database to handle performance efficiency, capacity, and resource utilization in a more efficient and effective way in the cloud environment. Here, resource utilization is optimum as nodes can be added based on requirement, and capacity also increases as increasing node can be done here easily.
Time behavior	Schema agnostic	NoSQL databases are schema agnostic. The great benefit of this is that development time is reduced as schema redesign takes little time compared to relational databases. The benefit increases if we need to go through multiple development releases and frequent alteration of internal data structure.
Availability Reliability Fault tolerance Recoverability	Distributed	The key advantage of distributed database is high availability, reliability, and achieving better fault tolerance to the user. Here, if data are replicated once or twice across other servers in the cluster, it will be easily accessible if one of the servers crashes or burns or affected due to natural disaster.

In the current world, as big data is growing at the annual rate of 60%, business needs a new solution to handle this large amount of data as the traditional database is not the optimum way to manage the situation. NoSQL is the new cloud database where service providers are developing more ways to host it in the cloud to get the maximum benefit. It is restricting its users from using their own resources while providing the ability to scale the databases into large capacities. [Table 5.2](#) shows how NoSQL performs better in the cloud computing environment due to its inherent aspects.

Bibliography

- Al Shehri, W. 2013. Cloud database database as a service. *International Journal of Database Management Systems*, 5(2), 1.
- Alam, B., Doja, M. N., Alam, M., and Mongia, S. 2013. 5-Layered architecture of cloud database management system. *AASRI Procedia*, 5, 194–199.
- Arora, I. and Gupta, A. 2012. Cloud databases: A paradigm shift in databases. *International Journal of Computer Science Issues*, 9(4), 77–83.

- Barker, B. 2015. Message passing interface (mpi). In *Workshop on High Performance Computing on Stampede*, Ithaca, New York.
- Bautista, L., Abran, A., and April, A. 2012. Design of a performance measurement framework for cloud computing. *Journal of Software Engineering and Applications*, 5(2), 69–75.
- Danwei, C. and Yanjun, H. 2010. A study on secure data storage strategy in cloud computing. *Journal of Convergence Information Technology*, 5(7), 23.
- Elzeiny, A., Elfetouh, A. A., and Riad, A. 2013. Cloud storage: A survey. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 2(4), 342–349.
- Eyal, I. 2013. *Scalable and Robust Algorithms for Cloud Storage and for Sensor Networks*. Technion-Israel Institute of Technology, Faculty of Electrical Engineering, PHD thesis—submitted on Sivan, Haifa.
- Geist, A. 1994. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA: MIT Press.
- Govinda, K. and Kumar, Y. 2012. Storage optimization in cloud environment using compression algorithm. *International Journal of Emerging Trends & Technology in Computer Science*, 1(1).
- Jadhav, M. S. P. and Nandwalkar, B. R. 2015. Efficient cloud computing with secure data storage using AES. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(6), 377–381.
- Jeong, H. and Park, J. 2012. An efficient cloud storage model for cloud computing environment. In: *Advances in Grid and Pervasive Computing, LNCS* (pp. 370–376), R. Li, J. Cao, and J. Bourgeois, Eds. Berlin, Heidelberg: Springer.
- Kalisch, M. and Bühlmann, P. 2007. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *The Journal of Machine Learning Research*, 8, 613–636.
- Kaur, R. and Kinger, S. 2014. Analysis of security algorithms in cloud computing. *International Journal of Application or Innovation in Engineering and Management*, 3(3), 171–176.
- Khedkar, S. V. and Gawande, A. D. 2014. Data partitioning technique to improve cloud data storage security. *International Journal of Computer Science and Information Technologies*, 5(3), 3347–3350.
- Khurana, N. and Datta, R. 2013. Logical data model for cloud computing. *International Journal*, 3(4), 937–943.
- Kwok, Y. K. and Ahmad, I. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4), 406–471.
- Patel, H. B., Patel, D. R., Borisaniya, B., and Patel, A. 2012. Data storage security model for cloud computing. In: *Advances in Communication, Network, and Computing* (pp. 37–45). Berlin, Heidelberg: Springer.
- Paul, M. and Das, A. 2015. A review on provisioning of services in cloud computing. *International Journal of Science & Research*, 3(11), 1–7.
- Raj, P. Ed. 2014. *Handbook of Research on Cloud Infrastructures for Big Data Analytics*. IGI Global, Hershey, PA.
- Rao, B. T. and Reddy, L. S. S. 2012. Survey on improved scheduling in Hadoop MapReduce in cloud environments. *International Journal of Computer Applications*, 34, 28.
- Salot, P. 2013. A survey of various scheduling algorithm in cloud computing environment. *International Journal of Research and Engineering Technology (IJRET)*, 2, 131–135.
- Singh, R. M., Paul, S., and Kumar, A. 2014. Task scheduling in cloud computing: Review. *International Journal of Computer Science and Information Technologies*, 5(6), 7940–7944.
- The Promise & Challenges of Cloud Storage—IMEX RESEARCH.COM. Available at http://www.imexresearch.com/newsletters/Aug10/Promise_of_Cloud/eNewsletter_Promise_& Challenges_of_Cloud%20Storage-2.pdf.
- Walker, M. B. 2012. NIST: Cloud reliability, information security remain 'open issues'. Available at <http://www.fiercegovernmentit.com/story/nist-cloud-reliability-information-security-remain-open-issues-cloud/2012-05-31>.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

6

A Scalable Record Linkage Technique for NoSQL Databases Using GPGPU

Parijat Shukla and Arun K. Somani

CONTENTS

6.1	Introduction	120
6.2	Background	122
6.2.1	NoSQL Databases	122
6.2.2	Definitions	122
6.2.2.1	Document Frequency	122
6.2.2.2	Levenshtein Distance	122
6.2.2.3	Similarity Metric	122
6.2.3	Hashing Table	123
6.2.4	General-Purpose Graphics Processing Units	123
6.2.5	Tree Data Model	123
6.3	Existing Methods	124
6.3.1	Stage Optimizations	124
6.3.1.1	Identifying Linkage Points	124
6.3.1.2	Blocking Mechanism	124
6.3.1.3	Similarity Techniques Optimizations	125
6.3.1.4	Hashing-Based Approaches	125
6.3.2	Cloud Computing-Based Approaches	125
6.3.3	Parallel Processing-Based Approaches	125
6.4	Record Linkage Using GPGPU	125
6.4.1	Stage 1: Preprocessing	126
6.4.1.1	Data Encoding and DF Update	127
6.4.1.2	Forming Signature Sets	127
6.4.2	Stage 2: Identifying Candidate Records	127
6.4.3	Stage 3: Linking Records	128
6.5	Signature Selection	128
6.5.1	Baseline Hashing-Based Approach	128
6.5.2	Signature Selector Data Structure: A Novel Approach	129
6.5.3	Forming Signature Sets	130
6.5.4	Discussion	131
6.5.4.1	Parallelization	131
6.5.4.2	Scalability	132
6.5.4.3	Collision Management	132
6.6	Parallel Algorithms for Signature Selection on GPGPU	132
6.6.1	Signature Set Selection Using a Lock-Based Approach	132

6.6.1.1	Extract Phase.....	132
6.6.1.2	Combine Phase.....	133
6.6.2	Lock-Free Signature Selection.....	133
6.7	Experimental Methodology	134
6.7.1	Experimental Platform.....	135
6.7.2	Data Sets.....	135
6.7.3	Metrics	135
6.8	Result	135
6.8.1	Performance of Hash-Based Approach for Data Encoding	136
6.8.2	Effect of Hash Table Size.....	136
6.8.3	Performance of Signature Selector-Based Data Pruning	137
6.8.4	F-Measure versus Memory Requirement.....	138
6.8.5	Overall Performance Comparison.....	138
6.9	Research Directions.....	139
	Acknowledgments	140
	References.....	140

6.1 Introduction

Big Data phenomenon has its impact on individuals, business, and governments alike. Analytics leveraging Big Data are key to drive business revenues, safety, and security in the society and improve quality of human lives. Core components in Big Data analytic domain need to be addressed diligently. However, such core components (in Big Data problems) require a tremendous amount of compute and memory, and networking resources.

Cloud computing (Armbrust et al. 2010) is a paradigm wherein resources such processor cores, memory, storage, and networking services are hosted on a giant computing infrastructures. Cloud computing services offer significant advantages to end user and businesses, alike in terms of lower cost, lower time to solution, pay as you go nature of services, etc. A cloud service provider manages the cloud service. Underlying large-scale computing infrastructures, also known as data centers, support cloud computing services. These data centers typically rely on commercial off-the-shelf (COTS) components such as commodity servers/processors, storage, and commodity network switches. Cost advantages offered by these commodity components are the prime reason for their prevalence in cloud computing segment. A recent trend is to leverage cost and power performance advantages of parallel architecture-based many-core processors such as general-purpose graphics processing units or GPGPUs (NVIDIAa 2016) and Intel’s many integrated cores architecture co-processors or MICs (Intel 2016).

Record linkage (Winkler 1995) is a core component of several Big Data analytic problems as well as key to database operations such as data integration and master data management. The record linkage problem is defined as follows. Given two sets of records, find whether a record belonging to one set corresponds to a record in the other set. Record linkage is important for integration of data from various sources and data sharing between organizations, analysis and mining of large data collections, e-commerce and web applications, etc. Specifically, record linkage problem is used for merging new records into a larger master data set, cleaning and enriching data for analysis and mining, detecting fraud, crime and security issues, and biomedical and social science research, etc.

Data of various formats, unstructured, structured, and semistructured, are being accrued at a humongous rates. Semistructured databases, such as document-oriented databases, are on the rise. The document-oriented databases comprise records, which are basically semistructured in nature. Semistructured data set, such as tree-structured ones, presents processing challenges due to their somewhat irregular forms. At the same time, semistructured data sets also offer significant advantages, namely there is no need to use a rigid schema, etc., over structured data representations. The compute-intensive nature of record linkage problem coupled with the semistructured nature of data found in document-oriented databases makes their processing challenging.

Record linkage problem obviously benefits from the parallel processing capabilities offered via cloud computing. Computations involving identification of linked records for a given record are independent. However, existing work addressing record linkage problem occurring in the domain of semistructured data does not leverage the parallel processing effectively. For example, none of the existing solutions make use of the enormous compute power of single-instruction multiple-thread (SIMT) machines.

Record linkage solution operates in multiple stages. The first stage is a data-pruning stage where records, which are not likely to result in a match, are pruned from further consideration. Data-pruning stage is followed by subsequent stage(s), after which the final output is generated. The data-pruning stage can be the most compute-intensive stage and oftentimes become a performance bottleneck. Effective application of parallel computing platforms such as GPGPUs for data-pruning and other stages of record linkage can deliver near real-time record linkage solution. However, using GPGPUs for linkage of semistructured data is not straightforward. The primary reason is heterogeneity. This presents challenges when considering GPGPUs, which operate in SIMT/data fashion. Also, selection of an appropriate data structure is crucial for designing a scalable record linkage solution considering parallel computing systems, for example, SIMT systems.

In this chapter, we present a SIMT-based approach to deliver near real-time record linkage solution in the context of document-oriented databases. To this end, we first combine a hashing-based data-shaping technique and information theoretic approach, and then we leverage compute-rich parallel architecture of SIMT-based machines. Our hashing-based technique has two-fold advantages—it reduces the space complexity involved in preprocessing stage and enables effective usage of hundreds of compute cores of SIMT machine to detect the linked records in a time-efficient manner.

Specifically, we make the following contributions:

- We develop a novel data structure for selection of useful signatures in the context of record linkage problem in document-oriented data sets on a GPGPU platform. We optimize the data structure using information theoretic approach.
- We explore efficient methods to compare signature sets on GPGPUs.
- We develop a scalable framework for linking records in document-oriented databases on a parallel machine.

Rest of this chapter is organized as follows. Section 6.2 covers the background relevant to this chapter. Section 6.3 details the existing methods. Section 6.4 presents our record linkage solution using databases. Sections 6.5 and 6.6 describe two specific components of our solution in detail. Sections 6.7 and 6.8 discuss the experimental methodology adopted and the results obtained, respectively. Conclusion and future research directions are highlighted in Section 6.9.

6.2 Background

In this section, we review terminologies and concepts relevant to this chapter. Specifically, we cover NoSQL databases and definitions on document frequency (DF) and distance metrics. We also provide a background on hash table and GPGPU, and tree data model.

6.2.1 NoSQL Databases

NoSQL (NoSQL 2016) databases are a special type of databases, which are characterized by the features such as being nonrelational, distributed, open source, and horizontally scalable. NoSQL databases are scheme free, easily replicable, simple to use, and able to handle a huge amount of data. NoSQL databases are especially useful when businesses need to access and analyze massive volumes of data, which is unstructured or semistructured in nature.

Popular NoSQL databases are Apache Cassandra, Simple DB, Google BigTable, MemcacheDB, etc. The NoSQL databases can be classified as wide-column store, document store, key-value store, graph databases, and others. Refer NoSQL (2016) for more details on NoSQL databases. A survey of NoSQL databases is provided in Han et al. (2011).

6.2.2 Definitions

In the following, we describe some commonly used terms.

6.2.2.1 Document Frequency

Document frequency (DF(t)) of a term t is defined as the total number of documents in a collection in which term t occurs. A related definition is Inverse Document Frequency (IDF). The intuition behind DF and IDF is that when a query term occurs in several documents, then that term is not a good discriminator. However, if a term occurs in fewer documents, then its discriminating power is higher and should be assigned a higher weight (or priority).

6.2.2.2 Levenshtein Distance

The Levenshtein distance is a common metric used for measuring the difference between two strings or sequences (Levenshtein 1966). Basically, the Levenshtein distance between two strings is the minimum numbers of single-character edit operations needed to change one string into another. Allowed edit operations are insertions, deletions, or substitutions. The Levenshtein distance is also referred to as string edit distance. Several variations of edit distance exist.

6.2.2.3 Similarity Metric

In a given data set, let D and D' be two semistructured data records. We define these records as linked if their similarity score $\text{Sim}(D, D')$ is greater than a predefined threshold value. A common formula to calculate similarity score is the following: $\text{Sim}(s_1, s_2) = [1 - \text{LevDist}(s_1, s_2)] / \text{len}(s_1)$, where $\text{LevDist}(s_1, s_2)$ is the Levenshtein distance between strings s_1 and s_2 , and $\text{len}(s_1)$ is the length of string s_1 .

6.2.3 Hashing Table

A hash table supports the basic dictionary operations such as insert, find, and delete. A given entry is inserted into the hash table as follows: obtain a hash value of the given entry using an appropriate hash function, and then write the given entry at the address given by hash function. A hash collision occurs when two (or more) keys map to the same hash value in the hash table. In the context of hash table, three design decisions are important: (1) size of the hash table, (2) hash function, and (3) collision resolution. The literature suggests several approaches to handle collisions in hash table. These approaches are classified into open addressing and closed addressing (Cormen 2009; Maurer and Lewis 1975). An example of open addressing is separate chaining. In separate chaining, all the keys that map to the same hash value (in the hash table) are kept in a list or a bucket. Examples of closed addressing are linear probing, quadratic probing, double hashing, perfect hashing (Majewski et al. 1996), cuckoo hashing (Pagh and Rodler 2001), etc.

6.2.4 General-Purpose Graphics Processing Units

GPGPUs (NVIDIAa 2016) are advancing the landscape of high-performance (high-throughput) computing operating in an SIMT mode. In GPUs, a large number of threads execute the same instruction concurrently. The GPUs comprise hundreds of streaming processor (SP) cores, which operate in parallel. SPs are arranged in groups, and each group of SPs is referred to as streaming multiprocessor (SM). All the SPs within one SM execute instructions from the same memory block in a lockstep fashion. GPUs are equipped with a large number of registers. Communication within SPs of one SM is through a low-latency shared memory. Main memory in GPUs is referred to as device global memory. Special purpose software-managed on-chip memories are provided on GPUs (NVIDIAb 2015). They include shared memory, constant memory (CM), and texture memory (TM) besides the regular L1/L2 cache memories.

6.2.5 Tree Data Model

In *tree data model*, labels are represented as nodes (NoSQL 2016; Ullman 2016). Refer to Figure 6.1. The tree model data set is encoded in XML (Quin 2013). Other encodings such

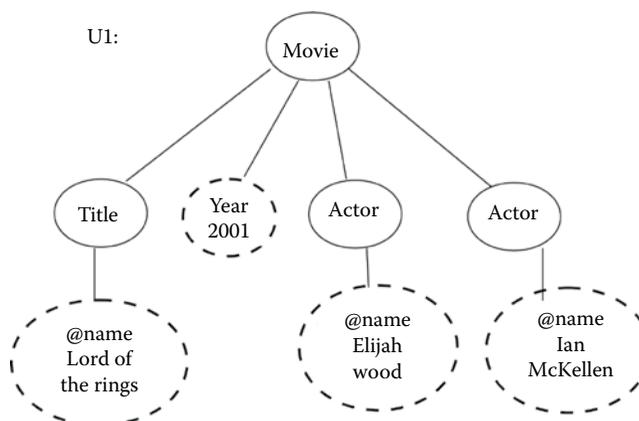


FIGURE 6.1

Tree U1. Only the leaf nodes, shown in dotted circle, are considered for signature sets.

as JSON (NA 1999) are also applicable. Labels in the tree model correspond to elements in the XML encoding. Elements are represented by an opening tag <element> and a closing tag </element>. Elements can have attributes and text (or content). Texts basically correspond to the unlabeled leaf nodes in the tree. The opening and closing tags delimit texts. Input data set comprises several tree model objects encoded in XML. This is a typical mode.

6.3 Existing Methods

Recent researches on record linkage problem occurring in semistructured data can be organized into the following three major categories: (1) micro-level optimizations or stage optimization approaches, (2) cloud computing approaches, and (3) parallel processing approaches. Table 6.1 depicts categorization of recent researches on record linkage problem occurring in semistructured data domain.

6.3.1 Stage Optimizations

Recent studies focusing on optimizations of the stage(s) in record linkage problem can be further categorized based on the particular stage where they are focused. Specifically, following subcategories are possible: identification of linkage points, blocking mechanisms, similarity techniques, and hashing-based approaches.

6.3.1.1 Identifying Linkage Points

Hassanzadeh et al. propose SMaSh framework, which discovers linkage point in a scalable, online manner. The framework operates in multiple stages: (1) a task scheduler, storage, and indexing backend; (2) registering and loading data sets; (3) analysis and indexing; and (4) search and discovery of linkage points.

A key component of their work is identification of linkage points in an efficient manner. This is realized by Hassanzadeh et al. (2013). Hassanzadeh et al. propose a framework comprising a library of efficient lexical analyzers and similarity functions, and a set of search algorithms for identification of linkage points over web data.

6.3.1.2 Blocking Mechanism

The entity resolution problem in real-time setting for structured data sets was studied in Ramadan and Christen (2014), which proposes a dynamic sorted neighborhood indexing

TABLE 6.1

Classification of Recent Research on Record Linkage in Semistructured Data

Category/Approach	Research
Stage optimization	Hassanzadeh et al. (2013), Ramadan and Christen (2014), Wang et al. (2011), Kim and Lee (2010)
Cloud computing based	Paradies et al. (2012), Papadakis and Nejd1 (2011)
Parallel processing based	Kirsten et al. (2010), Mamun et al. (2014), Sehili et al. (2015)

methodology. The indexing method proposed uses multiple trees with different sorting keys, which can be used for real-time entity resolution for read-most databases.

Papadakis and Nejd1 present an entity resolution method for heterogeneous information spaces (Papadakis and Nejd1 2011). Their focus was on developing novel blocking methods that scale up entity resolution within large, noisy, and heterogeneous information spaces. Their approach is attribute-agnostic and relies on values of entity profiles for building blocks.

6.3.1.3 Similarity Techniques Optimizations

Wang et al. studied similarity measures in the context of entity-matching problem (Wang et al. 2011). The study in Wang et al. (2011) characterizes various similarity measures used in entity matching and provides a solution to automate the selection process of appropriate similarity functions.

6.3.1.4 Hashing-Based Approaches

Kim and Lee propose HARPA (Kim and Lee 2010), which enables fast iterative-hashed record linkage for large-scale data collections. HARPA gains by dynamic and reusable hash table and exploitation of data characteristics.

6.3.2 Cloud Computing-Based Approaches

A cloud-based solution for entity matching in semistructured data is proposed in Paradies et al. (2012). The study in Paradies et al. (2012) combines ChuQL, an extension of XQuery with MapReduce and a blocking technique for entity matching.

6.3.3 Parallel Processing-Based Approaches

Kirsten et al. propose strategies to partition data for parallel entity matching (Kirsten et al. 2010). They also present a service-based infrastructure for parallel entity matching on different hardware settings. In order to reduce communication requirements, caching and affinity-based scheduling of match tasks are supported.

The study in Mamun et al. (2014) reports efficient sequential and parallel algorithms for linking records. The solution can handle any number of data sets. One recent work proposes a GPU-based solution for privacy-preserving record linkage for structured data (Sehili et al. 2015).

A comparison of the popular methods for entity matching is provided in Kpcke and Rahm (2010). A tutorial on the state-of-the-art in knowledge harvesting from web and text sources is provided (Suchanek and Weikum 2013). A survey of entity resolution and record linkage methodologies is provided in Brizan and Tansel (2015).

6.4 Record Linkage Using GPGPU

This section describes the process of record linkage in semistructured data sets on a GPGPU or GPGPU-like parallel hardware. We adopt a data-shaping-based approach, applying appropriate transformations to data and algorithm(s) involved in parallelizing

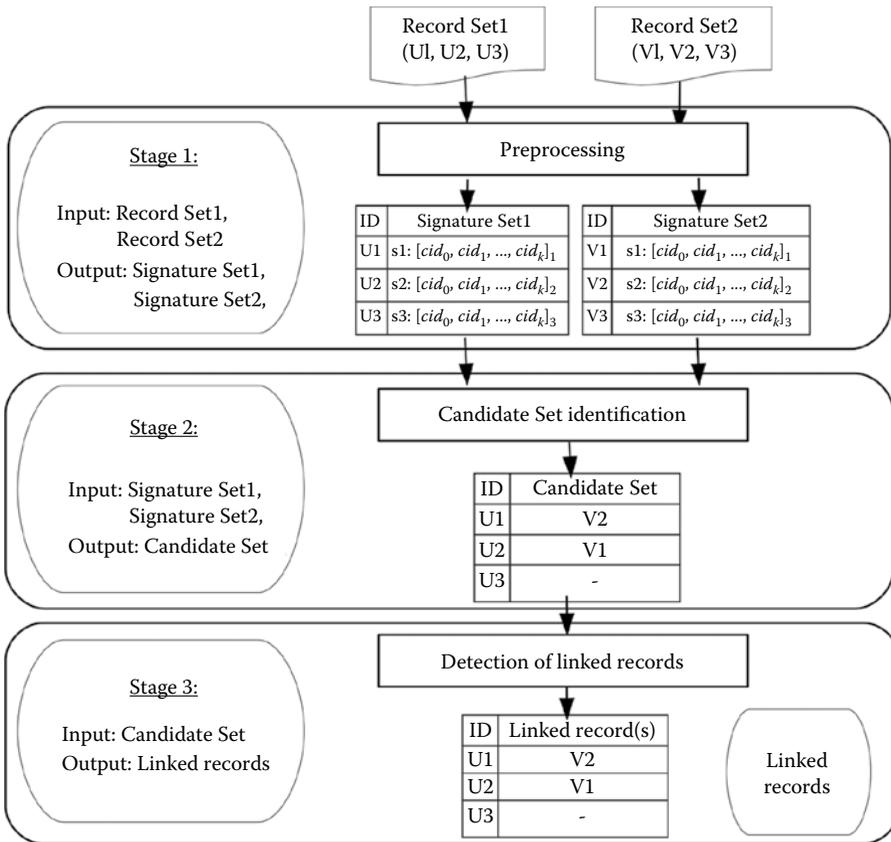


FIGURE 6.2 Framework for record linkage process. The figure shows the process using two sets of records: Record Set1 and Record Set2.

the computations. Parallelization of computations involved helps exploit the structured and rigid architecture of SIMT/SIMD-like parallel machines in an efficient manner.

The proposed record-linking solution operates in three stages: (1) preprocessing, (2) identification of candidate sets, and (3) linking records. Figure 6.2 illustrates the process flow.

6.4.1 Stage 1: Preprocessing

The terms that occur too frequently in the data sets do not help in identifying the linked records as it increases the probability of false positive. Similarly, infrequently occurring terms can increase the probability of false negatives in identification of linked records. Hence, it is prudent to use terms that are neither too frequent nor too rare when looking for linked records. DF, a robust searching technique from information theory, helps in defining relative occurrence of terms. We make use of DF to select useful terms, which later on helps prune the not-likely record candidates. Specifically, we use terms that fall in the prespecified range $\langle MIN\ DF, MAX\ DF \rangle$ for further search. Input to the preprocessing stage is an XML-encoded data set (comprising tree model objects). Output of the preprocessing stage is a record-wise signature set. Briefly, the preprocessing stage comprises the following.

6.4.1.1 Data Encoding and DF Update

Input data set is parsed using an appropriate parser (e.g., expat parser for XML-encoded data set). On encountering a leaf node, we update its DF. We do not consider the intermediate nodes (tags and textual values) when developing the signature sets for object.

6.4.1.2 Forming Signature Sets

For every object, select text terms, which fall in the prespecified range $\langle MIN_DF, MAX_DF \rangle$, and refer this as signature set of that object. A signature set tuple comprises *ContentID*. For example, signature set of the i th object comprising k -tuples is represented as $s_i = [cid_0, cid_1, \dots, cid_k]_i$ (Figure 6.3).

We implement Stage 1, that is, preprocessing stage using a hashing approach. Hashing approach helps avoid serious processing bottleneck. For example, with data encoding, there is no need for the costly string comparison-based search operations through the table storing the string terms and the numeric codes assigned to them. Moreover, using numeric codes instead of string values helps in storage and access as well.

Preprocessing stage that leads to overall efficient processing is the focus of this chapter. We describe the preprocessing stage in detail in Section 6.5.

6.4.2 Stage 2: Identifying Candidate Records

This is data reduction stage. Input to this stage is the list of object signature sets, and output is a set of candidate-linked records for that object. Algorithm used for identifying a set of candidate-linked records is listed as Algorithm 6.1.

Algorithm 6.1 takes a list of signature sets (S) as input. It compares signature set of all objects with each other (Line 6). Every tuple of set s_i is compared with every tuple of set s_j , and *Cand_Sim_Score* (indicating candidate similarity score) is calculated (Line 7). A match at any stage results in assigning a new *Cand_Sim_Score*, and further matching operations are not carried out for the tuples. If *Cand_Sim_Score* exceeds the candidate similarity threshold denoted by (θ) , object o_j is added to C_i , the candidate set of i th object (Lines 8–9).

Stage 2 is amenable to parallel processing. For example, this stage involves identification of candidate set for all the objects of Set1. Identification is single-process multiple-data kind of processing. Stage 2, therefore, can be and is executed on GPU.

ID	Signature set
U1	(The matrix), (L. Fishburne)
U2	(Matrix), (Keanu Reeves)
U3	(Lord of The Rings), (Peter Jackson)

FIGURE 6.3

Example of signature sets for three tree objects namely U1, U2, and U3. (Assume U2 and U3 similar to U1 shown above.) Note that actual signature set contains hashed values instead of text values.

Algorithm 6.1 IdentifyCandidateSet(S)

Input: $S = (s_0, s_1, \dots, s_{N-1})$, signature set of $N-1$ objects.
Output: $C = (C_0, C_1, \dots, C_{N-1})$, duplicate candidates for N objects.
 C_i : Candidate set for the i th object ($0 \leq i < N$).
Cand_Sim_Score: Candidate Similarity Score.
 θ_{cand} : Candidate Similarity Threshold.

```

1: for every  $s_i$  in  $S$  s.t.  $i \in [0, N)$  do
2:     for every  $s_j$  in  $S$  s.t.  $j \in [0, N)$  do
3:         if  $i \neq j$  then
4:              $C_i \leftarrow 0$ 
5:             Cand_Sim_Score  $\leftarrow 0$ 
6:             Compare  $s_i$  and  $s_j$  tuple-wise
7:             Assign Cand_Sim_Score
8:             if Cand_Sim_Score  $> \theta_{cand}$  then
9:                  $C_i \leftarrow C_i \cup o_j$ 
10:            end if
11:        end if
12:    end for
13: end for
```

ALGORITHM 6.1
Identify candidate set.

6.4.3 Stage 3: Linking Records

After Stage 2, for every object in Set1, we have identified a set of candidates belonging to Set2. In this stage, we refine the set of candidates for every object. Specifically, we carry out intensive pairwise comparison and identify the linked records. Input to this stage is the list of candidate set C , and output is a list of linked sets. The algorithm takes the list of candidate set C as input and produces a list of linked records for each object. We compare all nodes of o_i with all nodes of o_j , and LR_Sim_Score is calculated. If LR_Sim_Score exceeds the value of linked record similarity threshold denoted by (θ_{LR}) , object o_j is added to LR_i , the linked record set of i th object. Stage 3 is also parallelizable as it also involves a task that is essentially single process multiple data in nature. Stage 3 is also executed on GPU.

6.5 Signature Selection

In this section, we first review the concept of hash table data structure. Then, we discuss a naive approach based on hash table. Then, we describe our efficient hash-based approach for data encoding and candidate selection, the design rationales, and working. We also provide a discussion on the proposed hashing-based data structure.

6.5.1 Baseline Hashing-Based Approach

Hash table is a popular data structure. Typically, a hash table supports the basic dictionary operations such as insert, find, and delete. A given entry is inserted in the following

Algorithm 6.2 BaselineApproach(t)

```

T: A hash table of size S
1:  $h \leftarrow \text{Hash}(t)$ 
2: if ( $h > S$ ) then
3:      $h \leftarrow h \% S$ 
4: end if
5: Update the DF at address  $h$  in the  $T$ 
6: Update document ID list

```

ALGORITHM 6.2
Baseline approach.

way: obtain a hash value of the given entry using an appropriate hash function, and write the given entry at the address given by hash function. In the context of hash table, three design decisions are important: size of the hash table, hash function, and collision resolution. The literature suggests several approaches to handle collisions in hash table. These approaches are classified into open addressing and closed addressing (Cormen 2009; Maurer and Lewis 1975). An example of open addressing is separate chaining. Examples of closed addressing are linear probing, quadratic probing, double hashing, perfect hashing (Majewski et al. 1996), cuckoo hashing (Pagh and Rodler 2001), etc.

A naive approach to apply hashing mechanisms for preprocessing stage, that is, Stage 1 is shown in [Algorithm 6.2](#). The basic idea is as follows. Given a term t , obtain hash h of t . If h is outside the range of the hash table, then take a MOD of h and assign it to h . Update the DF count at address h in hash table. Update the document ID list.

The baseline approach discussed above has a serious limitation. Memory footprint of the hash table is prohibitively large and potentially a memory bottleneck. Recall that the record linkage problem in the era of Big Data must leverage the compute power of SIMT machines to address the processing time challenges. However, SIMT machines are limited in terms of memory. Thus, reducing the memory requirements of data structures and algorithms used to this end is critical to take advantage of SIMTs.

Reducing the size of hash table is an option, but it results in false collision leading to an artificial rise in the DF count of some terms. This effect, that is, the artificial rise in the DF count of some terms, can adversely affect the record linkage task by reducing the accuracy of records linked.

6.5.2 Signature Selector Data Structure: A Novel Approach

While a hash table is appropriate for dictionary operations (find, insert, and delete), the requirements of a data encoding and candidate selection are different. For example, the Signature Selector data structure (SignatureSel) need not perform any deletions. It only needs to carry out find and insert operations. The hash value collisions (or true collisions) affect the candidate selection process by artificially inflating the DF count of few terms. We also need to maintain an auxiliary data structure, which records the IDs of the documents (trees) in which the terms occur.

[Figure 6.4](#) depicts the design of our Signature Selector and its operation. [Figure 6.4a](#) shows that terms accompanying leaf nodes are considered for the signatures. It also shows that a hash value of such terms is generated using Hash() function. The hash value thus obtained is used as an entry in Signature Selector data structure. Since the size of Signature

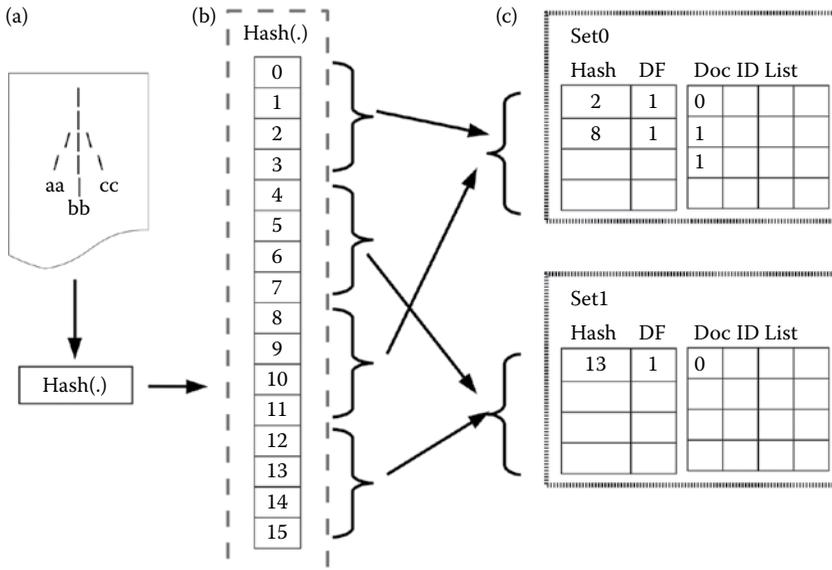


FIGURE 6.4 Signature Selector data structure. (a) A document and hashing of the leaf node terms. (b) The mapping of hash values to Signature Selector data structure (without hash table). (c) The organization Signature Selector into sets Set0 and Set1.

Selector is much smaller than the range of hash values obtained from Hash(), we need to map h to an appropriate line (and hash bucket) in Signature Selector. Mapping scheme is shown in Figure 6.4b.

Figure 6.4c shows the Signature Selector data structure. Signature Selector has three fields: (1) Hash which holds the hash of a given term, (2) DF which is the DF of the given term, and (3) List which contains all the documents in which the given terms occur. Note that, the size of the list, which contains document IDs, can become very large if the given term occurs in a large amount of documents. This is a potential problem. We address this issue in an intelligent manner. We conjecture that if a given term occurs in too many documents, its discerning power is limited, and hence its effectiveness to help identify potential candidate records (and potential linked records) is also very limited. Thus, it is prudent not to use such terms when identifying candidate records. Consequently, such terms have no relevance in the Signature Selector data structure. Hence, there is no need to maintain a list of documents for such terms. Formally, if the DF of a given term exceeds the MAX DF threshold, we do not maintain the list of documents in which the given term occurs.

Algorithm 6.3 describes the mapping and update procedure. For a given term t , we obtain hash h of t in Line 1. In Line 2, we derive the ID of hash bucket, bucketID, to which h is written into. In Line 3, we derive address (in bucketID) at which h is to be written. In Line 5, DF of h is incremented. If DF is below DF MAX threshold, the document ID list is updated (Lines 6 and 7).

6.5.3 Forming Signature Sets

For every object, we select text terms that fall in the prespecified range $\langle MIN\ DF, MAX\ DF \rangle$, and we refer this as the signature set of that object. A signature set tuple comprises

Algorithm 6.3 updateSignatureSel(t)

```

1:  $h \leftarrow \text{Hash}(t)$ 
2:  $\text{bucketID} \leftarrow \text{FLOOR}\left(\frac{h}{\text{numBuckets}}\right)$ 
3: if ( $\text{bucketID} > \text{numBuckets}$ ) then
4:      $\text{bucketID} \leftarrow \text{bucketID} \% \text{numBuckets}$ 
5: end if
6:  $\text{lineID} \leftarrow h \% \text{sizeBucket}$ 
7: Write  $h$  at line  $\text{lineID}$  in bucket  $\text{bucketID}$ 
8: Increment  $DF$ 
9: if ( $DF < \text{MAX\_DF}$ ) then
10:     Update document ID list
11: end if

```

ALGORITHM 6.3

Update signature selector.

Algorithm 6.4 buildSignaturesSeq(t)

T: A hash table of size S

```

1:  $H \leftarrow \text{Hash}(t)$ 
2: if ( $h > S$ ) then
3:      $h \leftarrow h \% S$ 
4: end if
5: Update the  $DF$  at address  $h$  in the  $T$ 
6: Update document ID list

```

ALGORITHM 6.4

Build signature set sequentially.

ContentID. For example, signature set of the i th object comprising k -tuples is represented as $s_i = [cid_0, cid_1, \dots, cid_k]$. Algorithm 6.4 depicts this process, which operates in a sequential manner on CPU. We discuss a parallel implementation of this process on GPU in Section 6.6.

6.5.4 Discussion

While designing the Signature Selector data structure, our goal is to enable a memory-efficient implementation. We do not consider linked list-based implementation because such an implementation is not efficient for GPGPUs. Moreover, our design favors efficient memory access while performing operations on Signature Selector data structure.

Specifically, our design considers the following aspects: (1) parallelization, (2) scalability, and (3) collision management.

6.5.4.1 Parallelization

Signature Selector data structure is amenable to parallel processing. The task of identifying relevant critical terms, that is, terms that belong to the range $\langle \text{MIN } DF, \text{MAX } DF \rangle$, can be done in parallel. All the hash buckets can be processed concurrently, thus utilizing the parallel processing capabilities of modern computing systems such as GPGPUs and MICs.

6.5.4.2 Scalability

Our design is scalable in terms of memory footprint. The hash-based candidate selector data structure reduces the size of memory footprint to a factor of “ $1/k$ ”. Hence, the size of data structure grows sublinearly in terms of the size of hash table.

6.5.4.3 Collision Management

Note that, the hash tables are subject to collisions. A collision occurs when the hash function produces the same hash value for two given distinct terms. We refer to such type of collision as a true collision. We believe that such collisions do not have adverse effect on the record linkage process. Collision can also be a result of folding (or aliasing) effect. Folding (or aliasing) occurs when the hash value of a term is outside the range of the hash table used and an MOD of the hash value needs to be used. We refer to such a collision as false collision. Since we use a hash range, which is large enough, we avoid aliasing effects.

6.6 Parallel Algorithms for Signature Selection on GPGPU

We discuss a set of parallel algorithms to extract signatures on GPU. Specifically, we discuss two approaches for extraction of signatures: (1) lock-based approach followed by a reduction to combine the results and (2) a lock-free approach.

6.6.1 Signature Set Selection Using a Lock-Based Approach

While designing a parallel solution for this process, we consider that the following constraint limits the memory footprint to the size of hash buckets used (refer to [Figure 6.4](#)) in the worst case. The process of building signature set on a parallel system, as described in [Algorithm 6.4](#), can be decomposed into two phases or parts: (1) extraction phase and (2) combine phase. In extraction phase, ContentIDs (i.e., hash value) present in hash buckets are arranged document wise in a parallel manner. For example, one SM extracts ContentIDs from one (or more) hash bucket(s) and writes into the corresponding list of the documents to which it belongs. This enables task parallelization. However, in this process, multiple SMs (or Blocks) have partial copies of the document list. These multiple copies need to be combined into a single copy in the combine phase. [Algorithm 6.5](#) lists the lock-based approach. The algorithm operates on multiple hash buckets in parallel. The algorithm operates in two phases described below.

6.6.1.1 Extract Phase

Note that in [Algorithm 6.5](#), it is likely that more than one thread may be trying to write to update the list of a document simultaneously. Reason being that in a given step two (or more) hash values have same document IDs in the DocIDList. Lines 7 and 8 are the critical region of code and must be protected by appropriate locking mechanisms to avoid race conditions and guarantee correctness of the result.

Algorithm 6.5 `extractSignaturesPar(B)`

```

B: hash bucket of size S
i: Integer
1: if threadID < S then
2:     if (hashDF[threadID] > MIN_DF) (hashDF[threadID] <= MAX_DF) then
3:         if (i < doc_count) then
4:             h ← hash[i]
5:             docid ← hashDocList[i]
6:             CRITICAL SECTION START:
7:             DocList[docid].[DocList[docid].count] ← h
8:             (DocList[docid].count) + +
9:             CRITICAL SECTION END:
10:        end if
11:    end if
12: end if

```

ALGORITHM 6.5

Extract signature sets in parallel.

6.6.1.2 Combine Phase

Note that Signature Selector data structure is processed in parallel by several blocks of threads on a GPU. For example, consider the following assignment: N sets namely, Set1, Set2, ..., Set($N-1$) are assigned onto N blocks, BLOCK0, BLOCK1, ..., BLOCK($N-1$) of GPU, respectively. This is a one-to-one mapping, that is, one set is assigned to one GPU block. Other mappings are also possible, for example, 2-to-1, 4-to-1, and 8-to-1 configurations wherein multiple sets are assigned to each block.

Each block produces a local version of signature set of all the records. These local versions contain partial signatures. These local versions need to be combined into a single copy using reduction. Two versions are combined into single one by a GPU block at each step. For example, in Stage 1, local versions produced by BLOCK0 and BLOCK1 are combined by BLOCK0. Similarly, versions of BLOCK2 and BLOCK3 are combined by BLOCK2. Finally, the local versions of BLOCK($N-2$) and BLOCK($N-1$) are combined by BLOCK($N-2$). Note that, half of the blocks such as BLOCK1, BLOCK3, ..., BLOCK($N-1$) are not assigned any work in Stage 1. The reduction process terminates after $\log(N) = M$ stages producing a complete signature set of all the records.

We observe that the lock-based approach (discussed above) has several shortcomings in the form of locking, excessive memory footprint, and finally a reduction phase. We do not pursue this approach. We explore design of an alternative approach for selecting signatures described below.

6.6.2 Lock-Free Signature Selection

[Algorithm 6.6](#) presents an algorithm based on the principle of output data ownership. This is a lock-free algorithm wherein there is no need to protect critical sections via locking mechanism. Basic idea is that the each thread takes ownership of collecting its signatures from all the hash buckets. Note that in each thread operates on the all hash buckets (Line 2). Lines 3–5 depict the looping constructs. The DF count of a hash value is obtained in Lines 6 and 7. Lines 8 and 9 determine if the DF of the hash value lies within the specified

Algorithm 6.6 extractSignaturesOutput-wise(B)

```

1:  if (blockIdx.x < NumBlocks) then
2:    if (threadIdx.x < threadsPerBlock) then
3:      for (i=0; i < NUM_HASH_BUCKETS; i++) do
4:        for j=0; j < HASH_BUCKET_SIZE; j++) do
5:          for (k=0; k<MAX_DF; k++) do
6:            lineid_hash_bucket=(i*HASH_BUCKET_SZ)+j;
7:            DF_count=DF_count1_ld[lineid_hash_bucket];
8:            if (DF_count>=MIN_DF) then
9:              if (DF_count<MAX_DF) then
10:                hash_val=Hash_Bucket1_ld[lineid_hash_bucket];
11:                t_doc_id=List_DOC_ID1_ld[(i*HASH_BUCKET_SZ*MAX_DF)+(j*MAX_DF)+k];
12:                if (t_doc_id>0) then
13:                  if (t_doc_id==((block:Idx.x)*(threadsPerBlock)+threadIdx.x)) then
14:                    t_cnt=doc1_list_sel_list_count[(blockIdx.x)*(threadsPerBlock)+
                      (threadIdx.x)];
15:                    if (t_cnt<MAX_CAND) then
16:                      (doc1_list_sel_content_id[((blockIdx.x)*(threadsPerBlock)+
                      (threadIdx.x))*
                      (MAX_CAND)+t_cnt]=hash_val);
17:                      (doc1_list_sel_list_count[(blockIdx.x)*(threadsPerBlock)+
                      (threadIdx.x)])++;
18:                    end if
19:                  end if
20:                end if
21:              end if
22:            end if
23:          end for
24:        end for
25:      end for
26:    end if
27:  end if
28:  __syncthreads();
   //Process for second dataset:
29:  __syncthreads();

```

ALGORITHM 6.6

Output-wise signature selection in parallel.

range. In Line 10, the hash value is obtained from the hash bucket. A list of documents in which this hash value occurred are accessed in Line 11. Line 12 is provisioned for sanity check. In Line 13, the GPU thread verifies if the k th document is in the list belongs to itself or not. In other words, if the document id and the thread id are a match, then the thread will update its list of signatures (Lines 14–17). Same procedure is repeated for the second data set (not shown in the figure in interest of space).

6.7 Experimental Methodology

In this section, we describe the experimental setup and data sets used, and metrics used for performance comparison.

TABLE 6.2

Description of Data Sets

Data Set	Number of Records in (Set1,Set2)	Average Size of Record (KB)
Nasa	1K-2K, 1K-2K	11.5
Swissprot	1K-2K, 1K-2K	12.39

6.7.1 Experimental Platform

For experiments, we used a CPU/GPGPU-based platform. CPU is an Intel (R) Xeon (R) CPU E5-2650 @ 2.00 GHz machine running GNU/Linux. The GPU used is a Kepler K20 device comprising 2496 Cuda cores or SPs @ 706 MHz and is equipped with 5 GB GDDR5 on-board memory. All the algorithms are implemented in C (for CPU) and Cuda version 7.5 (for GPGPU).

6.7.2 Data Sets

We experiment using two real data sets, Nasa and Swissprot. Nasa contains geographic data, and Swissprot contains bioinformatics data. These data sets are obtained from UW XML database and are described in [Table 6.2](#).

From these real data sets, we construct following data sets for the purpose of record linkage experiments: (1) Nasa1K+1K and Nasa2K+2K and (2) Swissprot1K+1K and Swissprot2K+2K.

Nomenclature is explained here: Nasa1K+1K indicates that records from Nasa data set with Set1 and Set2 each having 1K records. Similarly, Nasa2K+2K indicates that records from Nasa data set with Set1 and Set2 each having 2K records. Same holds for Swissprot.

6.7.3 Metrics

We measure the effectiveness of record linkage approach using three metrics: precision, recall, and F-measures. Precision measures the percentage of correctly identified linked records (duplicates) over the total set of records determined as linked one (or duplicates) by the system. $\text{Precision} = (\text{TP})/(\text{TP} + \text{FP}) = \text{TP}/(\text{declared duplicates})$.

Recall measures the percentage of linked records (duplicates) correctly identified by the system over the total set of linked records (duplicate). $\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) = \text{TP}/(\text{true duplicates})$.

F-measure is defined as harmonic mean of precision and recall. $\text{F-measure} = (2 \times \text{recall} \times \text{precision})/(\text{recall} + \text{precision})$. We measure the execution time elapsed using Linux "time" command.

6.8 Result

We study the following: performance of hash-based approach for data encoding, effect of the size of hash table on accuracy of linking records, performance of signature selector-based data pruning, tradeoff between F-measure versus memory requirements of the hash-based methods, and overall impact of using GPU implementation of our hash-based approach. Now, we discuss the results obtained.

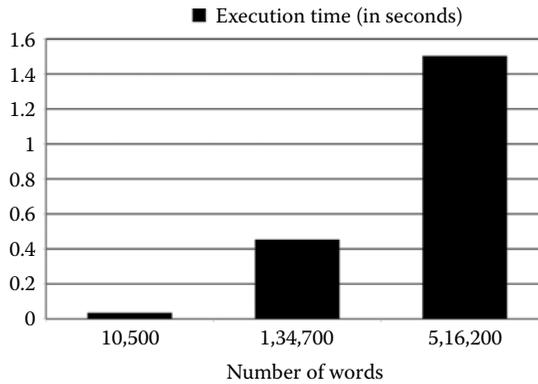


FIGURE 6.5
Runtime of hash functions.

6.8.1 Performance of Hash-Based Approach for Data Encoding

Figure 6.5 depicts the implication of using a hash-based approach for assigning numeric codes to string terms. X-axis is the number of words whose hash value is calculated. Y-axis is the time elapsed in hashing step. The figure shows a linear relationship between the elapsed time in hashing and the number of words hashed as expected.

6.8.2 Effect of Hash Table Size

Figure 6.6 depicts the effect of varying the size of hash table used in the record linkage process for Nasa. X-axis shows the size of hash table used in multiples of 1K (1024). Y-axis shows the total number of records identified. The figure shows that hash table of 64K is sufficient for finding the linked records for data sets comprising 1K+1K records. The figure also shows that hash table of 64K entries is sufficient for finding the linked records for data sets comprising 2K+2K records.

Figure 6.7 depicts the effect of varying the size of hash table used in the record linkage process for Swissprot. X-axis shows the size of hash table used in multiples of 1K (1024).

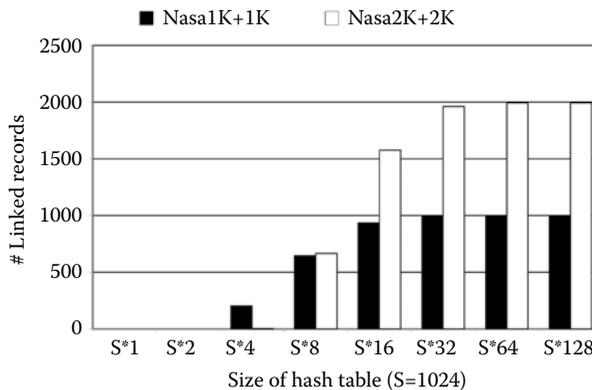


FIGURE 6.6
Effect of the size of hash table on accuracy for Nasa data set.

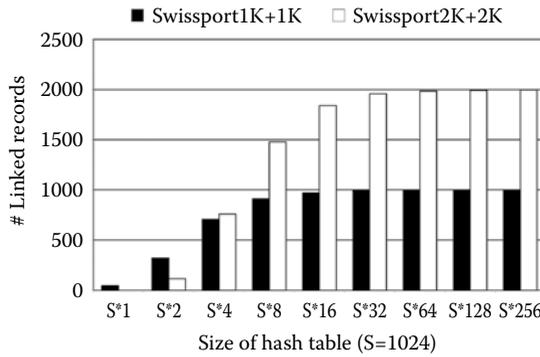


FIGURE 6.7
Effect of the size of hash table on accuracy for Swissprot data set.

Y-axis shows the total number of records identified. The figure shows that hash table of 128K is sufficient for finding the linked records for data sets comprising 1K+1K records. The figure also shows that hash table of 256K is sufficient for finding the linked records for data sets comprising 2K+2K records.

6.8.3 Performance of Signature Selector-Based Data Pruning

Now, we discuss effect of varying the number of hash buckets on the accuracy in linking records. Figure 6.8 plots the effect of the number of hash buckets on the accuracy of record linkage for Nasa1K+1K data set. X-axis shows the number of hash buckets each of size 64 entries varying from 32 to 2048 sizes. Y-axis shows the total number of records identified. From this figure, we observe that a 128 hash buckets each of size 64 yield an acceptable accuracy for the record linkage problem. With 128 hash buckets, we observe an insignificant number of false positives, 0.81%. Increasing the number of buckets to 256 or more yields approximately similar performance.

Figure 6.8 also depicts the effect of varying the number of hash buckets for Nasa2K+2K data set. From this figure, we observe that a 256 hash buckets each of size 64 yield an acceptable accuracy for the record linkage problem. Using 256 hash buckets yields 1.61% false positives, which is insignificant in the context.

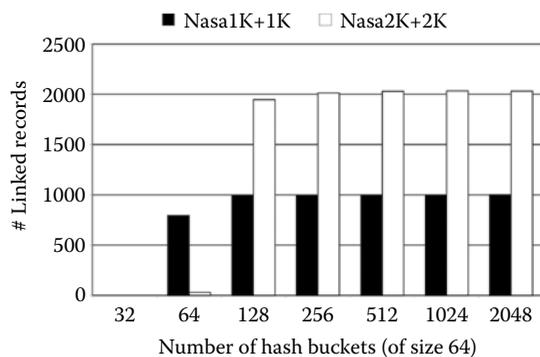


FIGURE 6.8
Effect of the number of hash buckets on the accuracy of record linkage for Nasa data set.

6.8.4 F-Measure versus Memory Requirement

Figure 6.9 compares the effect of memory needs on the accuracy of baseline approach and our signature selector data structure. The figure depicts the effect of memory usage on F-measure. X-axis shows the number of memory requirement in terms of entries in hash table of signature selection data structure varying from 1024 to 1024×256 . Y-axis shows the F-measure obtained. Note that F-measure is the harmonic mean of precision and recall metrics.

From Figure 6.9, we note that for Nasa1K+1K data set, baseline hash-based approach (Section 6.5.1) requires a hash table of size 32K for an acceptable accuracy level (i.e., accuracy >0.99). And, our optimized signature selector-based approach (Section 6.5.2) requires 128 hash buckets of 64 entries (equivalent to 1024×8 entries), each are sufficient to keep false-positive rate within an acceptable limit. This indicates that the use of Signature Selector data structure results in a reduction of 4X in memory requirement at the preprocessing stage.

Similarly, for Nasa2K+2K data set, the baseline approach, hash-based approach, requires a hash table of size 32K, and our optimized Signature Selector-based approach requires 128 hash buckets (of 64 entries each) for an acceptable accuracy level. This indicates that the use of Signature Selector data structure reduces the memory requirement of preprocessing stage by 4X.

6.8.5 Overall Performance Comparison

A table-based implementation to identify signature sets in the context of duplication detection problem is discussed in Shukla and Somani (2014). Here, we discuss the overall performances of record linkage implementation utilizing the table-based implementation for CPU against our novel hashing-based implementation for GPU, which leverages lock-free signature selection algorithm.

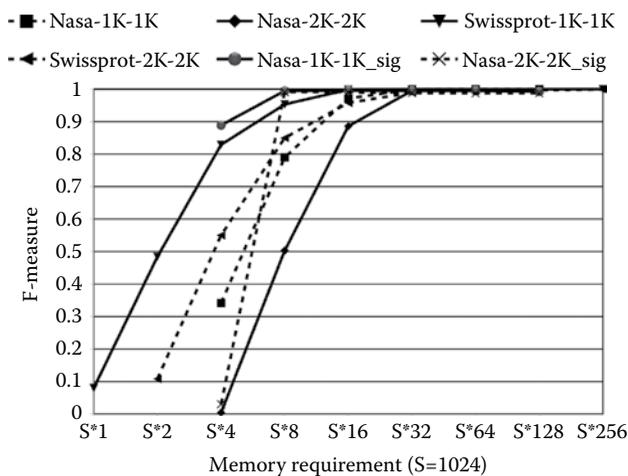
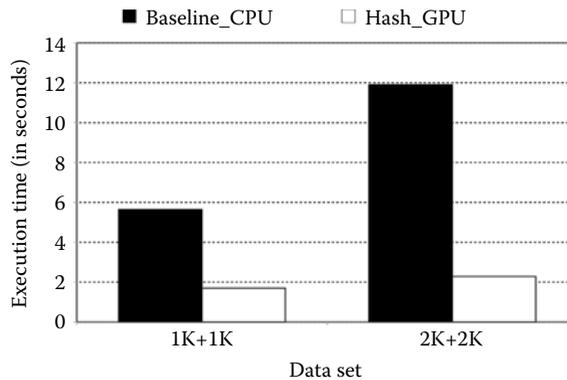


FIGURE 6.9

Effect of the size of hash table on the accuracy for Nasa data set.

**FIGURE 6.10**

Performance of baseline record linkage method on CPU (depicted as Baseline_CPU) versus hashing-based method on GPU (depicted as Hash_GPU).

Figure 6.10 depicts a comparison of overall execution time of a CPU implementation of a table-based record linkage method (based on Shukla and Somani [2014]), depicted as Baseline_CPU, versus the hashing-based method on GPU, depicted as Hash_GPU. X-axis shows the type of data set used in measurement, specifically, the Nasa1K+1K and Nasa2K+2K data sets. Y-axis shows the end-to-end time, in seconds, elapsed in execution of the algorithms on the respective platforms. For GPU implementation, we used the following configuration parameters: `threadsPerBlock = 128`, `HASH_BUCKET_SZ = 64`, and `NUM_HASH_BUCKETS = 128`.

From Figure 6.10, we note that for Nasa1K+1K data set, table-based approach (Shukla and Somani 2014) takes, approximately, 5.64 s. Our hash-based method approach (Section 6.6.2) requires, approximately, 1.707 s to produce the linked records. This indicates that the use of Signature Selector data structure when implemented using lock-free approach results in a speedup of, approximately, 3.3X.

Similarly, for Nasa2K+2K data set, the table-based approach (Shukla and Somani 2014) takes, approximately, 11.898 s. Our hash-based method approach (Section 6.6.2) requires, approximately, 2.286 s to produce the linked records, resulting in a speedup of, approximately, 5.2X. Note that despite the increase in speedup from 3.3X to 5.2X, the record linkage workload has been doubled. This indicates that our hash-based algorithms designed for GPUs are scalable.

6.9 Research Directions

This work suggests several directions of research. One of the directions possible is to explore design space to address the record linkage problem occurring in unstructured data domain leveraging the research discussed in this chapter. Another research direction is to explore the applicability of similar hash-based algorithms for efficient mining of linked records in semistructured and unstructured data sets using parallel architecture-based processors.

Acknowledgments

The authors would like to thank Mayank Mishra and Piyush Lakhawat for their detailed comments and for holding several rounds of insightful discussions. The authors would also like to acknowledge the funding agencies. Research is funded in part by the Jerry R. Junkins Endowment and Philip and Virginia Sproul Professorship endowment at Iowa State University. The research implementation reported in this paper is partially supported by the HPC@ISU equipment at Iowa State University, some of which has been purchased through funding provided by NSF under MRI grant number CNS 1229081 and CRI grant number 1205413. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G. et al. 2010. Above the clouds: A Berkeley view of cloud computing. *Communications of the ACM* 53(4): 50–58.
- Brizan, D. G. and Tansel, A. U. 2015. A survey of entity resolution and record linkage methodologies. *Communications of the IIMA* 6(3): 41–50.
- Cormen, T. H. 2009. *Introduction to Algorithms*. Cambridge, MA: MIT Press.
- Han, J., Haihong, E., Le, G., and Du, J. 2011. Survey on NoSQL database. *2011 6th International Conference on Pervasive Computing and Applications (ICPCA)*. Port Elizabeth: IEEE, 363–366.
- Hassanzadeh, O., Pu, K. Q., Yeganeh, S. H., Miller, R. J., Popa, L., Hernández, M. A., and Ho, H. 2013. Discovering linkage points over web data. *VLDB Endowment* 6(6): 445–456.
- Intel. 2016. *Intel Xeon Phi Product Family*. <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>.
- Kim, H.-s. and Lee, D. 2010. HARRA: Fast iterative hashed record linkage for large-scale data collections. *Proceedings of the 13th International Conference on Extending Database Technology*. ACM, 525–536.
- Kirsten, T., Kolb, L., Hartung, M., Groß, A., Köpcke, H., and Rahm, E. 2010. Data Partitioning for Parallel Entity Matching. *arXiv preprint arXiv:1006.5309*, 2010.
- Kpcke, H. and Rahm, E. 2010. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering* (Elsevier) 69(2): 197–210.
- Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10(8): 707–710.
- Majewski, B. S., Wormald, N. C., Havas, G., and Czech, Z. J. 1996. A family of perfect hashing methods. *The Computer Journal* 39(6): 547–554.
- Mamun, A.-A., Mi, T., Aseltine, R., and Rajasekaran, S. 2014. Efficient sequential and parallel algorithms for record linkage. *Journal of the American Medical Informatics Association* (BMJ Publishing Group Ltd) 21(2): 252–262.
- Maurer, W. D. and Lewis, T. G. 1975. Hash table methods. *ACM Computing Survey (CSUR)* 7(1): 5–19.
- NA. 1999. *Introducing JSON*. <http://www.json.org>. Accessed June 1, 2016.
- NoSQL. 2016. *NoSQL*. <http://nosql-database.org>. Accessed June 7, 2016.
- NVIDIAa. 2016. *What is GPU Computing?*. <http://www.nvidia.com/object/what-is-gpu-computing.html>. Accessed June 6, 2016.
- NVIDIAb. 2015. *Cuda Toolkit Documentation*. <http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4AvElewc2>. Accessed June 1, 2016.

- Pagh, R. and Rodler, F. F. 2001. Cuckoo hashing. In *European Symposium on Algorithms* (pp. 121–133). Springer Berlin Heidelberg.
- Papadakis, G. and Nejdil, W. 2011. Efficient entity resolution methods for heterogeneous information spaces. *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*. IEEE, 304–307.
- Paradies, M., Malaika, S., Siméon, J., Khatchadourian, S., and Sattler, K.-U. 2012. Entity matching for semistructured data in the cloud. *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 453–458.
- Quin, L. 2013. *Extensible Markup Language (XML)*. <http://www.w3.org/XML/>. Accessed June 1, 2016.
- Ramadan, B. and Christen, P. 2014. Forest-based dynamic sorted neighborhood indexing for real-time entity resolution. *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*. ACM, 1787–1790.
- Sehili, Z., Kolb, L., Borgs, C., Schnell, R., and Rahm, E. 2015. Privacy preserving record linkage with PPJoin. *Proceedings of BTW Conference* (pp. 85–104), Hamburg, Germany.
- Shukla, P. and Somani, A. K. 2014. Context-aware duplicate detection in semi-structured data streams. In *2014 IEEE World Congress Services (SERVICES)*. Anchorage: IEEE, 216–223.
- Suchanek, F. and Weikum, G. 2013. Knowledge harvesting in the big-data era. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 933–938.
- Ullman, J. D. 2016. *The Tree Data Model*. <http://infolab.stanford.edu/~ullman/focs/ch05.pdf>. Accessed June 6, 2016.
- Wang, J., Li, G., Yu, J. X., and Feng, J. 2011. Entity matching: How similar is similar. *Proceedings of the VLDB Endowment (VLDB Endowment)* 4(10): 622–633.
- Winkler, W. E. 1995. *Matching and Record Linkage*. www.census.gov. <https://www.census.gov/srd/papers/pdf/rr93-8.pdf>. Accessed 6 June 2016.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

7

NoSQL for Handling Big and Complex Biological Data

Awanish Kumar

CONTENTS

7.1	Introduction	143
7.2	Biological Database	145
7.2.1	Introduction	145
7.2.2	Biological Data Domain and Mining	146
7.2.3	Types of Biological Database	147
7.2.4	Some Important Biological Databases	147
7.3	Biological Database Management System	148
7.3.1	Introduction	148
7.3.2	Models Used in DBMS	149
7.3.2.1	Hierarchical DBMS	149
7.3.2.2	Network	149
7.3.2.3	Object Oriented	149
7.3.2.4	Relational	150
7.3.3	Languages in DBMS	150
7.3.4	Challenges	152
7.3.5	Why NoSQL in Biology	153
7.4	Big Data and NoSQL in Different Areas of Biological Research	154
7.4.1	Clinical Research	155
7.4.2	Genomic Research	155
7.4.3	Proteomic Research	156
7.4.4	Others	156
7.5	Implementation	157
	References	157
	Websites	158

7.1 Introduction

Biological data are very big, complex, and unstructured. The NoSQL has emerged from programmer aversion to strict physical data models with widespread perception that such models to accommodate Big Data volumes. To manage data in and extract information from a key-value store, you have to write code to walk the data structures. Unstructured biological data have significant benefits to honoring that structure in the storage layer especially at Big Data volumes. With the turn of the century, the technological revolution, and the onset of social media, Big Data has become critical to our everyday lives.

By enabling us to make advancements in genomics, medicine, proteomics, and other biological research, it has helped us find, store, deliver, and analyze large volumes of distributed data. Most interesting, perhaps, is Big Data's role in genomics and its ability to identify, map, and sequence hundreds of millions of strands of DNA. After the completion of Human Genome Project, a Big Data in terms of DNA sequence of the human genome was acquired which was required to analyze. These findings helped scientists effectually analyze the association between genetics and disease. Today, our society is one step closer to a reality in which we can predict diseases and use genomics data to create personalized, data-driven medicine. We are approaching a future in which statistically correlated data will have the ability to analyze each patient's unique DNA accurately and effectively, empowering doctors to assign personalized treatment plans to their patients.

In order to achieve this medical breakthrough, a massive amount of data are needed to be stored, analyzed, and understood. So, we have to ask ourselves, would it be possible for us to map and sequence the DNA for each and every person on the planet without the use of a robust Big Data management for knowledge (Cameron, 2014). Information retrieved from Big Data always enhances knowledge (Figure 7.1). To ensure this resilience, a comprehensive information technology can be used to monitor big data. A trend analysis can also be done and reported in real time which is necessary. This cross-domain correlation can only be effectively accomplished via a unified monitoring platform that can collect all the key metrics, end to end, and then present real-time views of complex biological data. The basic motivation of NoSQL was to make it easy to build and deploy applications. It makes easy to scale and operate vast biological data by having a distributed design. It can be more cost effective for the technological and medical breakthrough, which will likely revolutionize biological data in healthcare for serving the world.

In this book chapter, we describe the effectiveness of NoSQL in big biological data management and interpretation. The technology/technical terms used in the book chapter are explained wherever they appear or at the "Key Terminology & Definitions" section. Apart from regular References, additional References are included in the "References for Advance/Further reading" for the benefit of advanced readers.

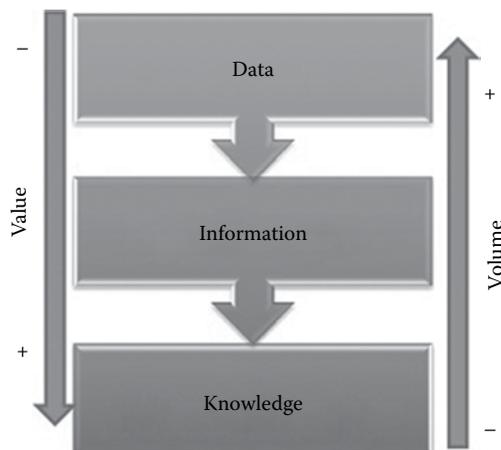


FIGURE 7.1

A workflow from data to knowledge.

7.2 Biological Database

7.2.1 Introduction

Now, the field of biological science has turned into a data-rich science. Therefore, building of biological databases is necessary to manage this biological information. A biological database is a collection, storage, and organization of data so that its contents can easily be accessed by the end users and updated. It is a highly convenient method for collection of a vast amount of information, and it allows for proper searching, storing, and retrieving of data. It supports large-scale analysis efforts, makes data access and update easy, and finally links the knowledge obtained from various fields of biology and medicine. Most of the biological databases have a web interface to search. The end user's common mode of search is using keywords. Users can choose to view the data or save the data on their computer. Cross-references help navigate from one biological database to another very easily. Biological databases represent a precious resource in support of research related to biological and other fields. We can learn much about a particular molecule by searching biological databases and using available analysis tools. A large number of biological databases are available on web. Some databases are very general, while others are very specialized. For the best results and knowledge, we should access multiple databases. Common biological database search methods include matching of keyword, searching of class, sequence, motif, etc. The problems with using biological databases include data spread over multiple databases, redundant and incomplete information, constant change, and various errors. Sometimes, database standards and naming are not clearly defined for many aspects of biological information. It makes information extraction of data more difficult.

A biological database is a large and organized corpus of persistent data. It is usually associated with computerized software designed to update and retrieve components of the stored data within the system. A simple biological database might be a single file containing many biological records, each of which includes the same set of information which gives the birth of bioinformatics and data mining. Bioinformatics and data mining are now developed as an interdisciplinary science. Mining biological data helps to extract useful knowledge from massive biology datasets.

For example, a record associated with a nucleotide sequence database typically contains information such as contact name; input sequence of DNA/RNA; scientific name of the source organism from which an organism was isolated; and, often, literature citations associated with a particular sequence. At present, a lot of bioinformatics work is concerned with the biological databases. These databases include both "public" repositories and "private" databases. Making such databases accessible via open standards like the web is very important since users of bioinformatics data use a range of computer platforms. Macromolecules like DNA and RNA store the hereditary information of an organism. These macromolecules have a fixed structure, which can be analyzed by biologists with the help of bioinformatics tools and databases. Because of the high speed of biological data production and the need for researchers to have rapid access and analyze, biological databases have become the major medium of research, and its development and management are continuously required as it is an essential source of information nowadays. Therefore, demand for database development and services is continually escalating in the biological community.

7.2.2 Biological Data Domain and Mining

Domain means a specific field or discipline. Before search or use of biological database, first we have to decide on which domain we have to search. Protein–protein interaction, DNA–protein interaction, inhibitor design, etc. are a few examples of domains on which we can focus. Data domain refers to all the information/values of a particular data in a data management and database analysis. The rule for determining the boundary of domain may be as straightforward as a data type with a specific list of values. Search of targeted data domain will make the study easy. Domain surfing is very important in biological database for the search of particular or targeted information. Domain selection/identification makes the database search easy and narrow down the problems. Evolution of domain knowledge saves time in research.

Data mining is the process of analyzing different perspectives of data from database and summarizing it into useful information. It allows users to analyze data from different dimensions/angles to categorize and summarize particular relationships identified. Data mining software is one of a number of analytical tools used for analyzing data. Broadly, data mining is the process of finding correlations or patterns among various fields in large databases. Biologists are stepping up their efforts in understanding the different biological processes. This has resulted in a flood of biological data from genomics to proteomics and from environmental to clinical data. A huge amount of data are available from DNA microarrays, protein sequences, protein interactions, biomedical images, disease pathways, and other health records (Xiaoli et al., 2014). These data can be exploited for discovering new knowledge and can be translated for health care. But there are difficulties in fundamental data analysis that have to be overcome. There are practical issues associated with the use of these databases such as handling incomplete and noisy data, processing tasks, and integrating various data sources. These new challenges are faced by biologists in the twenty-first century, but these can be overcome by data mining. Various data mining techniques have been designed to handle such challenging problems in data analysis and have been demonstrated with real applications. Data mining enables biologists and clinicians to make meaningful observations and discoveries from a wide array of heterogeneous biological data. Data mining provides the link between large-scale information transaction and analytical systems. Data mining software analyzes relationships and patterns in stored transaction data according to end user queries. Several types of analytical software are available: machine learning, neural networks, and statistical (Han and Kamber, 2006). In general, there are four types of relationships: Classes, Clusters, Associations, and Sequential patterns. Data mining has five major components: (a) storage and management of data in a multidimensional database system; (b) extraction, transformation, and transaction of data; (c) provision of data access to end user; (d) analysis of data by application software; and (e) presentation of data in a practical format, such as a graph or table.

Different levels of analysis in data mining are possible with the help of following methods/tools:

- *Artificial neural networks*: It is a nonlinear predictive model that learns through training, and structurally, it resembles biological neural networks.
- *Data visualization*: It is a visual interpretation of complex biological relationships in multidimensional data with the help of graphics tools.
- *Decision trees*: It generates rules for the classification of a dataset. These are tree-shaped structures that represent sets of decisions. Specific decision tree methods include Classification and Regression Trees and Chi-Square Automatic Interaction

Detection, both of which are used for classification of a dataset. They provide a set of rules that can be applied to a new/unclassified dataset to predict the outcome.

- *Genetic algorithms*: It is an optimization technique that uses genetic combination, mutation, and natural selection on the concepts of natural evolution.
- *Rule induction*: The extraction of useful data from a database is based on its statistical significance.
- *Nearest neighbor method or k-nearest neighbor technique*: This technique classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset (where $k \geq 1$).

Today, data mining applications are available for all sizes of data, and these tools are used to analyze and interpret biological data frequently and significantly.

7.2.3 Types of Biological Database

Vast information is available in the field of biological research. Therefore, biological databases are necessary, and they are developed for miscellaneous purposes. It encompasses various types of data at heterogeneous coverage and curates at different levels with different methods so that there are different criteria applicable to database classification accordingly. Biological databases contain the information of bibliography, literature, taxonomic, classification of nucleic acid, genomics, proteomics, gene/protein families, domains and functional sites, diseases, etc. Based on these huge information, different types of biological database have been created, and they are categorized as follows:

1. *Primary databases*: Experimental results are directly uploaded into primary database; therefore, it is also known as database of derived data. It contains sequence data such as nucleic acid or protein. Examples of primary databases include Protein Databases, SWISS-PROT, TREMBL, and PIR.
2. *Secondary databases*: It contains results from the analysis of the sequences in the primary databases; therefore, sometimes it is known as pattern databases. Examples of secondary databases include PROSITE, Pfam, BLOCKS, and PRINTS.
3. *Composite databases*: It is a combination of different sources of primary databases. It makes querying and efficient searching without the need to go to each of the primary databases. Examples of composite databases include NRDB (Non-Redundant DataBase) and OWL.

7.2.4 Some Important Biological Databases

The present challenges are to handle a huge volume of data such as those generated by the human genome project, develop and improve biological database design, develop software for database access and manipulation, and develop device data entry procedures to compensate for the varied computer procedures and systems used in different laboratories. Biological databases play a vital role in therapeutics and other applications. They offer an opportunity to user to access a wide variety of relevant data, including the gene/protein sequences of a broad range of organisms. Based on this information, we can also categorize biological databases as sequence databases (GenBank, the UCSC Genome Browser, Ensembl) and model organism databases (WormBase, the Arabidopsis Information Resource [TAIR], Mouse Genome Informatics [MGI]). A nonsequence-centric

database is another category that includes Online Mendelian Inheritance in Man (OMIM), the Protein Data Bank (PDB), MetaCyc, and the Kyoto Encyclopedia of Genes and Genomes (KEGG). Apart from these, a number of biological databases are available, and they are primary nucleotide sequence databases, meta databases, radiology databases, genome and proteome databases, protein sequence and structure databases, protein model and function databases, protein–protein and other molecular interactions, RNA and carbohydrate databases, signal transduction and metabolic pathways databases, microarray and PCR databases, taxonomic and phenotypic database, and some specialized databases.

7.3 Biological Database Management System

7.3.1 Introduction

Due to increase of research activities, there is production or generation of a huge amount of data in the recent past. Therefore, management of these valuable data is rapidly becoming a major determinant, and there is an urgent challenge to organize all biological sequences so that the biological information is optimally accessible. Therefore, a successful management system was developed known as Database Management System (DBMS), and the purpose of DBMS is to store, extract, and modify the data for end users (Figure 7.2).

It manages database proliferation, more and more scientific discoveries that result from inter-database analysis and mining, and rising complexity of required data combinations.

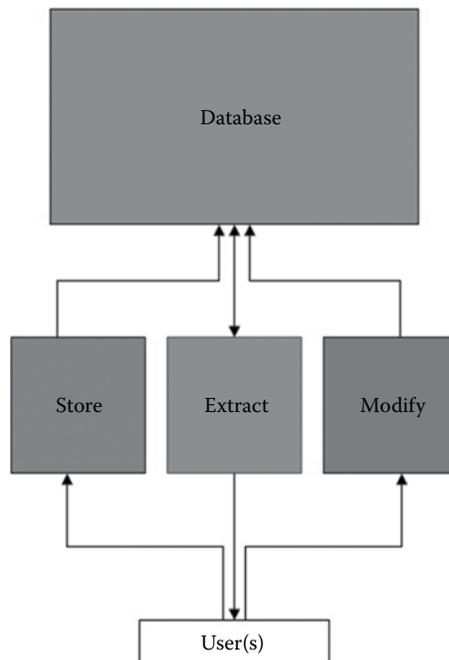


FIGURE 7.2
Process and purpose of DBMS.

7.3.2 Models Used in DBMS

Logical design of data is done by a particular model in database. The model depicts the relationships between different parts of the data. Various models have been developed to create and manage biological database. They are explained in the following sections.

7.3.2.1 Hierarchical DBMS

It is simple and restrictive. This model systematizes data in a tree structure (Figure 7.3). There is a hierarchy in parent and child data segments. It has only one root record. Each root record may participate in relationship with many child records. A parent record owns its child records, and if the parent record is deleted, all child records are automatically removed. Changing the structure of the database prepared based on this model is very complicated because changes in structure need changes in the access mechanisms and consequently in the programs of application.

7.3.2.2 Network

This model is a collection of related types of record. There is no restriction on the number and direction of links that can be established. This model was developed in response to the limitations of the hierarchical model. There is no root record, and each record can participate in any number of own relationships (Figure 7.4). In this model, duplication of data is eliminated. It is possible to set up record instances without having them participate in a link and by deleting one owner. This model does not necessarily delete all its other members. However, implementation of this model is highly obscured.

7.3.2.3 Object Oriented

It is a complex and versatile model. This model adds database functionality to object programming languages. It brings much more constant storage of programming language objects. It has full-featured database programming potential. A major benefit of this model is the unification of the application and database development into a seamless data model

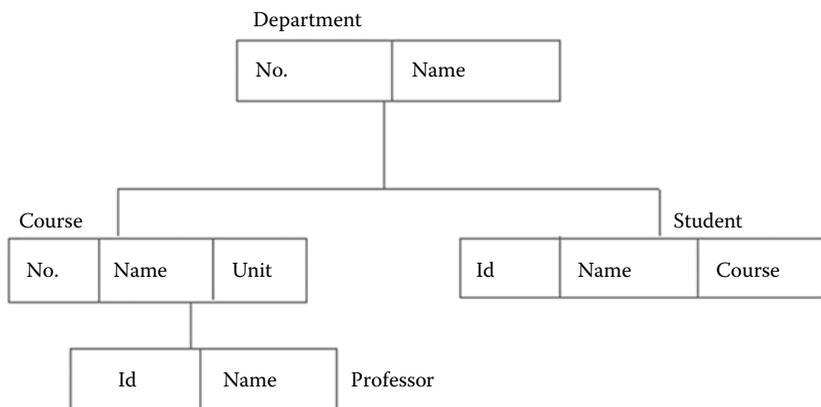


FIGURE 7.3
Schematics of hierarchical DBMS.

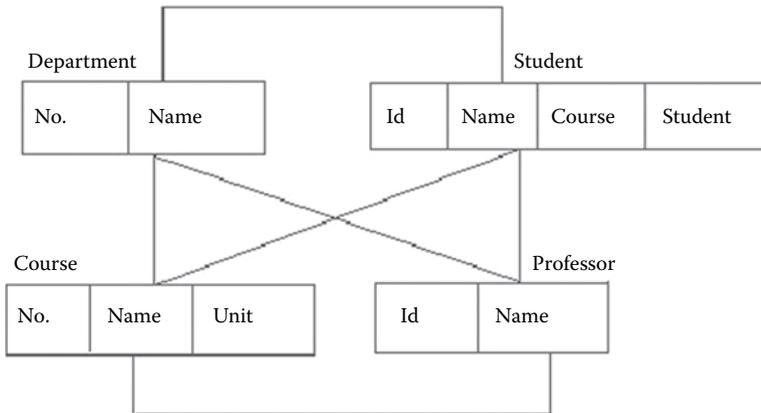


FIGURE 7.4
Schematics of network DBMS.

and environment of language. As a result, applications require less code, and code bases are easier to maintain with a modest amount of additional effort.

7.3.2.4 Relational

It is also complex, versatile, and in tabular form. Here, data are presented as a collection of relations. Each relation is represented as a table. Columns represent attributes, and rows are entities. Every table has a set of attributes that taken together as a key uniquely identifies each entity (Figure 7.5).

7.3.3 Languages in DBMS

Standard and interactive programming languages are required in DBMS for updating and organizing a database. Two or more data files in different locations are periodically

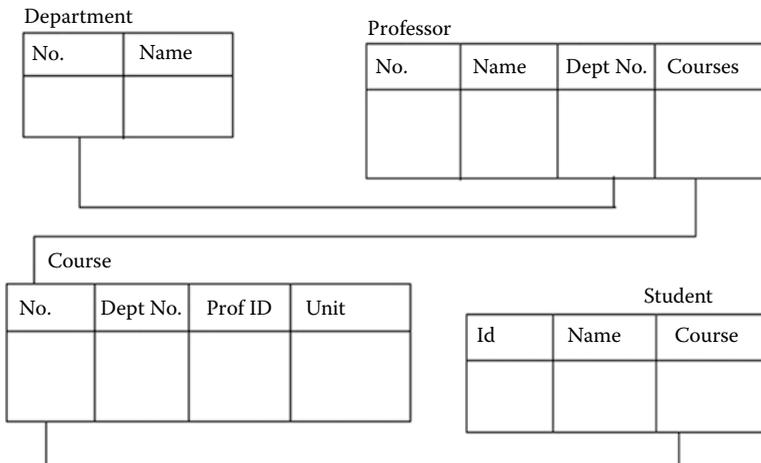


FIGURE 7.5
Schematics of relational DBMS.

synchronized by the management system to keep data in all locations regularly available to the users.

1. *Data Definition Language (DDL)*: It allows a database administrator or database designer to define tables, create views, etc. The functions of DDL are as follows:
 - To create the database instance
 - To alter the structure of database
 - To drop database instances
 - To delete tables in a database instance
 - To rename database instances
2. *Data Manipulation Language (DML)*: It allows an end user to retrieve information from tables. DML is of following types:
 - a. *High-Level or Nonprocedural Language*: It is “set” oriented and specifies what data to retrieve rather than how to retrieve them. They are also called as declarative languages.
 - b. *Low-Level or Procedural Language*: It is defined as retrieve data one record at a time. It creates a looping that are needed to retrieve multiple records along with positioning pointers.

It is used for accessing and manipulating data:

 - i. To read records from table
 - ii. To insert record into the table
 - iii. Update the data in table
 - iv. Delete all the records from the table
3. *Data Control Language (DCL)*: It is used for granting/revoking user access on a database:
 - To grant access to the user
 - To revoke access from the user
4. *Structured Query Language (SQL)*: It is also called as SEQUEL (Structured English QUery Language). It came from an IBM Research project entitled SEQUEL, where the intent was to create a structured English-like query language to interface to the early System R database system. It is used to converse with any database. It is a standard language for relational DBMS (RDBMS) according to American National Standards Institute. SQL statements are used to do tasks such as update data of a database or retrieve data from a database. SQL uses Sybase, Oracle, Microsoft SQL Server, Access, and Ingres for database development and management. Although most of the databases employ SQL, most of them also have their own additional proprietary extensions. However, the standard SQL commands such as Create, Select, Insert, Update, Delete, and Drop can be used to accomplish almost everything that one needs to do with a database.
5. *Object-oriented DBMS (OODBMS)*: It is the combination of object-oriented programming (OOP) principles with database management principles. OOP includes encapsulation, polymorphism, and inheritance they are enforced for database management concepts on the basis of ACID properties (Atomicity, Consistency, Isolation and Durability). It leads to system integrity, support for ad hoc query

language, and secondary storage management systems which allow for managing very Big Data sets. The object-oriented database manifesto specifically lists many features (complex objects, object identity, types and classes, encapsulation, overloading and late binding, overriding, class or type hierarchies, persistence, computational completeness, extensibility, secondary storage management, concurrency, an ad hoc query facility and recovery) as mandatory for a system to support.

OODBMS is able to store objects that are nearly identical from the kind of objects supported by the target programming language with little limitation. Persistent objects should belong to a class and can have one or more types of objects as attributes. Each object has an object identifier (OID), which is used as a way of uniquely identifying a particular object. OIDs are permanent and system generated. It is not based on any of the member data within the object. OIDs store references to other objects in the database but may cause referential integrity problems if an object is deleted while other objects still have references to its OID. An OODBMS is thus a full-scale, object-oriented development environment as well as a DBMS.

6. *Relational DBMS (RDBMS)*: It is a program that lets us to create, update, and administer a relational database. Most of the RDBMS use the SQL to access the database, although SQL was invented after the development of the relational model (Lee et al., 2013). Some features are common in the RDBMS and OODBMS such as indexes, deadlock detection, the ability to handle large amounts of data, transactions, data recovery mechanisms, and backup and restoration features. It was introduced in the 1970s. RDBMS avoided the navigation model as in the old DBMS and introduced relational model. The relational model has relationship between tables using primary keys, foreign keys, and indexes. Thus, getting and storing of data become faster than the old navigational model. So, RDBMS is widely used by enterprises and developers for storing complex and Big Data.
7. *Not Only SQL (NoSQL)*: It is an approach to management of data and database design that is highly useful for extremely large sets of distributed data. It encompasses a wide range of architectures and technologies that search and solve the scalability and Big Data performance issues that a relational database is not designed to address. NoSQL is especially useful when a venture needs to access and analyze huge amounts of unstructured data. It is also used for Big Data management of data that are stored remotely on multiple virtual servers in the cloud. NoSQL does not forbid any SQL. While it is true that some NoSQL are entirely nonrelational, others simply avoid selected relational functionality. Instead of using tables, a NoSQL database might organize data into objects, value–key pairs. Therefore, NoSQL is often mentioned in conjunction with other Big Data tools, for example, columnar-based databases, massive parallel processing, and Database as a Service. This database model evolved recently (Figure 7.6) and brought many benefits to the computer industry (B.M.e., 2014).

7.3.4 Challenges

Managing Big Data is a challenging task. Big Data technologies are maturing to a point where most of the organizations adopt Big Data as a core component of the information analysis and management. Big Data is positioned as a vast step in enabling integrated analytics in different sub-areas of biological science (Wang et al., 2014). With the advent of

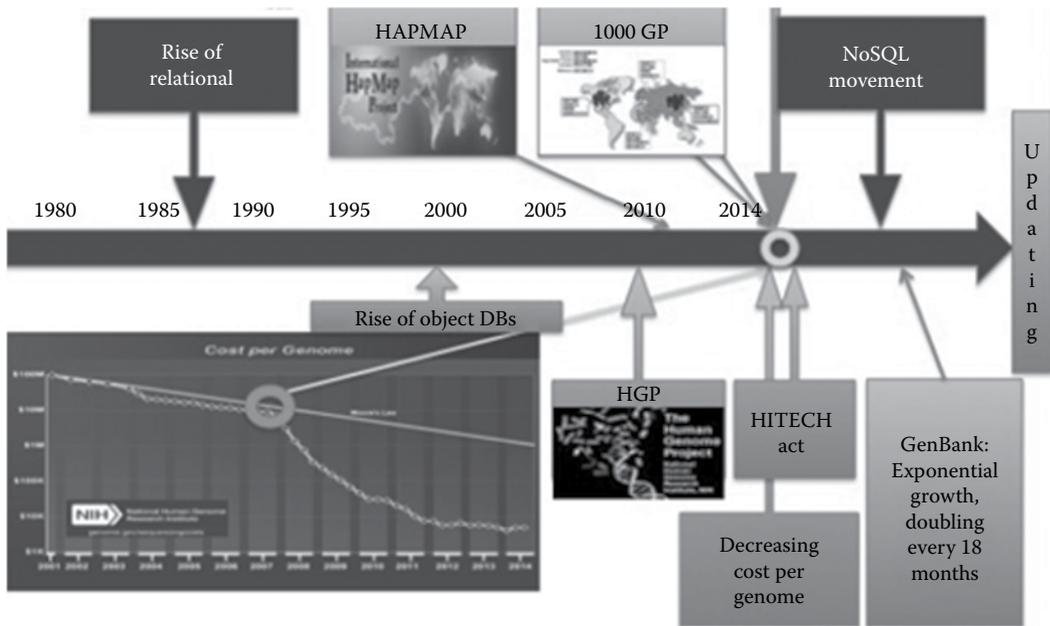


FIGURE 7.6
Timeline of DBMS.

high-throughput techniques, biologists are starting to struggle with massive data sets and encountering challenges in handling, processing, and extracting meaningful information. A number of challenges are associated with big biological data. It could be as follows:

1. Generation of huge raw data
2. Uncertainty of the data management
3. Competing technologies
4. Big gap between data generation, analysis, and implementation
5. Ease of data accessibility and integration our third challenge
6. Synchronization
7. Getting meaningful information out of the huge data platform
8. Storage and augmentation

To overcome these issues, researchers have to explore how a strategy for data integration can be crafted to manage risks of these problems.

7.3.5 Why NoSQL in Biology

The reasons for biology to adopt a NoSQL database over a relational database are the growth and generation of Big Data. Big Data is one of the key forces driving the growth and popularity of NoSQL in biology. The almost limitless array of data collection technologies ranging from simple *in vitro* study to *in vivo* and clinical study act as force multipliers for data growth in biological science. In fact, one of the first reasons to use NoSQL is that

biological science has a Big Data project to tackle. A biological study is normally typified by the following:

1. *High data velocity*: Lots of data are coming very quickly from different parts of the world as a pilot-scale study.
2. *Data variety*: Biological data are generated in different varieties like structured, semistructured, and unstructured.
3. *Data volume*: Acquired data are in terabytes or more in size.
4. *Data complexity*: Biological data that are stored and managed in different locations or data centers are very complex.

These complexities and vastness can be overcome and managed by NoSQL comfortably because NoSQL has features such as workload diversity, high performance, scalability, strong manageability, and continuous availability.

7.4 Big Data and NoSQL in Different Areas of Biological Research

Powerful computers and numerous tools for complex biological data analysis are crucial in proteomics, genomics, drug discovery, and other areas of Big Data biology, that is, in various categories (Figure 7.7) (SAS, 2011). Each category presents distinct data model characteristics, operational aspects, drivers for adoption, and functional capabilities. If 10 or more genomes are analyzed for comparison, then hundreds or more computational steps

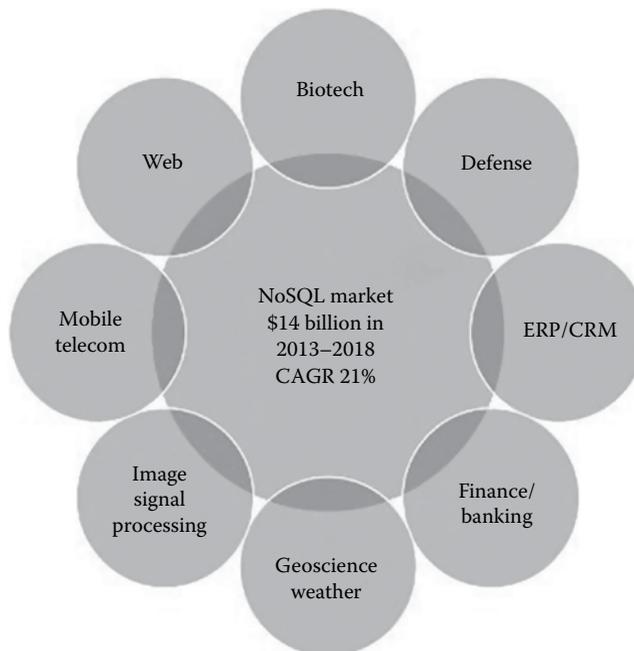


FIGURE 7.7

Application of NoSQL in different field of research (www.cloudave.com).

are involved. There are numerous NoSQL technologies that have become popular in managing Big Data sets comfortably. These technologies can be categorized as follows: document model, key-value model, column model, and graph model, and these technologies are implemented in different areas of biology where big and complex data are generated (Li and Chen, 2014).

7.4.1 Clinical Research

Clinical data are vast, sporadic, dynamic, and heterogeneous in nature, and their logical analysis and successful implementation for diagnosis are necessary. Special attention should be paid to the design of clinical database due to its unique features. Large clinical data research studies are becoming ubiquitous and offer a number of potential benefits (Shoenbill et al., 2014). Each research study needs to be considered carefully in its own right, together with the justification for using the data for that specific purpose. Currently, storage of clinical data mainly relies on RDBMS which is the most common and a proven approach to store and query data in various forms (Atzeni and Antonellis, 1993). However, the major drawback of this system is the need to predesign the exact field structures of the data, which is required in the process of database normalization to ensure data uniformity. In addition, the RDBMS is not practical. A relational database storing these kinds of data will contain many empty fields, resulting in inefficient storage and poor performance. To deal with these issues, there is requirement of NoSQL to manage and develop clinical databases that can cope with the special features of clinical data more effectively. Clinical databases have multiple levels of nesting, and the complete data model that is compiled can be complicated. Hence, one should consider NoSQL databases as an option. Multilevel nesting and hierarchies are very easily symbolized in the JavaScript Object Notation (JSON) format used by some document-oriented NoSQL products. Document-oriented NoSQL data store documents that are not required to have the same number of fields, nor must all documents be of the same basic structure. NoSQL databases scale out, and they do not require a fixed predefined schema definition. And also, if there is a change in data capture and management needs, NoSQL provides an easy way to handle it as it allows freely adding the fields to JSON documents without having to first define changes. The format of the data being inserted can be changed at any time, without disruption the application. This allows application developers to move quickly to incorporate new data into their applications. MUMPS (Massachusetts General Hospital Utility Multi-Programming System), ClinVar are the vast clinical database which is based on NoSQL. It stores data in multidimensional hierarchical sparse arrays for the user and is highly applicable in diagnosis of patients. Every year, 100 or more megabits of data are generated and stored in electronic health records (EHR) of hospitals, and experts expect that this rate will increase as genetic data expand over time. NoSQL technology can address big clinical data challenges such as those faced in the medical field. Continuous progress has been made in determining the advantages of NoSQL approaches for managing clinical and genomic data.

7.4.2 Genomic Research

As genomic technology advances (especially with the onset of Next Generation Sequencing), data of genomic information is becoming exponentially bigger and more complex, which requires breakthrough algorithmic and computational solutions (Canuel et al., 2015). As a result, "Big Data Research and Development Initiative" was announced in 2012 by the United States. The focus was kept on improving the ability to manage large amounts of

genomic data and excerpt new knowledge. However, this initiative is just the beginning in addressing the management challenges presented by the huge volumes of digital data produced by genome-based research. Thus, the future work will need to aim at making dramatic advances in data management systems, together with the development of tools and techniques needed to store, analyze, organize, access, and assemble massive collections of genomic data in terms of digital information that will help in genomic medicine practice (Mayer et al., 2011). The Cancer Genome Atlas (TCGA), Pittsburgh Genome Resource Repository (PGRR), HUGO, and DNP databases are based on NoSQL for effective data storage and management in precision medicine.

7.4.3 Proteomic Research

The age of research after year 2000 was known as post genomic or proteomics era. The term “proteome” refers to the study of entire complement of proteins, including the modifications made to a particular set of proteins produced by an organism at cellular level. It varies with time and distinct requirements, such as stresses, stimuli, treatment, and other environmental conditions. The term “proteomics” refers to a large-scale comprehensive study of a specific proteome, including protein abundances, their variations, and modifications, along with their interacting partners and networks. The greatest promise for the detection and treatment of any disease lies in the deep understanding of molecular basis for disease initiation, progression, and efficacious treatment based on the discovery of unique proteins as biomarkers. Although progress in genomics has been rapid during the past few years, it only provides us with a glimpse of what may occur as dictated by the genetic code. In reality, we still need to measure what is happening in a patient in real time in terms of proteins that provide real insight into the biological processes of disease development. This is because genes are only the “recipes” of the cell, while the proteins encoded by the genes are ultimately the functional players that drive both normal and disease physiology. Due to its high importance, research in proteomics is accelerated, which will generate Big Data very fast. Scientists have been looking for a better approach than the predominant but inefficient relational model to manage proteomics data. At this point in time, NoSQLs, with their potential to deal with large, heterogeneous, and even dynamic databases in proteomics, are gaining attention. As mentioned above, NoSQL databases were originally created by web developers to revolutionize the management of real-time data, with which SQL databases had many problems in management. The use of NoSQL has become popular because, unlike SQL, NoSQL does not require strict schemas and keys, for example, HUPO, mass spectrometry database.

7.4.4 Others

The concept of NoSQL is also introduced to bioinformaticians for their potential benefits. Over the past decade, researchers encountered problems in bioinformatics analysis, where they realized that graph-friendly databases were needed. In fact, the indexing step itself can take very long time for a large graph. All that the bioinformatics community needs to do is to leverage on an existing infrastructure. Thus came the NoSQL to overcome these problems because graph-friendly NoSQL databases exist. The Big Data challenge is also found in bioinformatics. Data storage and processing, instead of experimental technologies, are becoming the slower and more costly part of biological research. Biological data typically have large size and a variety of structures. The ability to efficiently store and retrieve the data is important in bioinformatics research. Traditionally, large datasets are

either stored as disk-based flat files or in relational databases. These systems become more complicated to plan, maintain, and adjust to Big Data applications as they follow rigid table schema and often lack scalability. Nonrelational databases (NoSQL) emerge to provide alternative, flexible, and more scalable bioinformatics data storage and management. There is no doubt that bioinformatics will have a significant impact on biological sciences and betterment of human lives.

7.5 Implementation

When compared to relational databases, NoSQL databases are more scalable and provide superior performance. Therefore, implementation of NoSQL is high, and it addresses several issues that the relational model is not designed to address. Large volumes of rapidly changing structured, semistructured, and unstructured data are successfully managed. This is because a relatively small number of software developers are familiar with NoSQL approaches. The key-value model implemented in NoSQL databases holds promise to be more performant solutions. Medicine, clinical, and genomic data require frequent updates because every day there are new scientific findings which are frequently not predefined in a database that consequently requires schema changes. These types of issues are difficult for SQL databases to address. Since managing high write loads and schema flexibility are very expensive in terms of processing and operational spends, making changes to the database after implementing these features can further increase costs. Scalability, dynamics, structural change, and updating process are key aspects in NoSQL databases that may advantage in the manipulation of big and dynamic structural data over SQL systems. Many researchers have proposed that NoSQL technology holds a great promise for large-scale data management, in particular for high-dimensional biological data such as that demonstrated in the performance evaluation (Stonebraker, 2010; Wang et al., 2014). Leaving SQL and implementing NoSQL is turning out a great win in the development and management of big and complex biological data. Therefore, nowadays, NoSQL databases are really a choice for persisting big and complex data.

References

- Atzeni, P. and Antonellis, V. D. 1993. *Relational Database Theory*. San Francisco, CA: Benjamin-Cummings Publishing.
- B.M.e. 2014. Clinical database: RDBMS v/s newer technologies (NoSQL and XML database); why look beyond RDBMS and Consider the newer. *International Journal of Computer Engineering and Technology (IJCET)*, 5, 73–83.
- Cameron, D. 2014. *Transforming “Big Data” into Knowledge*.
- Canuel, V., Rance, B., Avillach, P., Degoulet, P., and Burgun, A. 2015. Translational research platforms integrating clinical and omics data: A review of publicly available solutions. *Briefings in Bioinformatics*, 16, 280–290.
- Han, J. and Kamber, M. 2006. *Data Mining Concepts and Techniques*. Morgan Kaufmann (Elsevier). ISBN: 673, 978-0-12-381479-1.

- Lee, K. K., Tang, W. C., and Choi, K. S. 2013. Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage. *Computer Methods and Programs in Biomedicine*, 110, 99–109.
- Li, Y. and Chen, L. 2014. Big biological data: Challenges and opportunities. *Genomics, Proteomics & Bioinformatics*, 12(5), 187–254.
- Mayer, A. N., Dimmock, D. P., Arca, M. J., Bick, D. P., Verbsky, J. W., Worthey, E. A., Jacob, H. J., and Margolis, D. A. 2011. A timely arrival for genomic medicine. *Genetics in Medicine*, 13, 195–196.
- SAS. 2011. How big is ‘big data’ in healthcare? Available at: <http://blogs.sas.com/content/hls/2011/10/21/how-big-is-big-data-in-healthcare/>.
- Shoenbill, K., Fost, N., Tachinardi, U., and Mendonca, E. A. 2014. Genetic data and electronic health records: A discussion of ethical, logistical and technological considerations. *Journal of the American Medical Informatics Association: JAMIA*, 21, 171–180.
- Stonebraker, M. 2010. SQL databases v. NoSQL databases. *Commun Communication of the ACM*, 53(4), 10–11.
- Wang, S., Pandis, I., Wu, C., He, S., Johnson, D., Emam, I., Guitton, F., and Guo, Y. 2014. High dimensional biological data retrieval optimization with NoSQL technology. *BMC Genomics*, 15(Suppl 8), S3.
- Xiaoli, L., See-Kiong, N., Jason, T., and Wang, L. 2014. Biological data mining and its applications in healthcare. *Science, Engineering, and Biology Informatics*, 8, 436, ISBN: 978-981-4551-00-7.

Websites

www.cloudave.com.

<http://www.omim.org>.

<http://www.rcsb.org/pdb/home/home.do>.

<http://www.genome.jp/kegg>.

8

Applications of Hadoop Ecosystems Tools

Pratik Mishra, Mayank Mishra, and Arun K. Somani

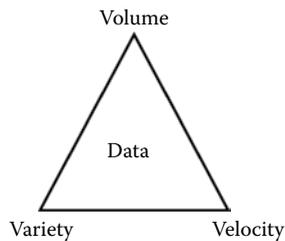
CONTENTS

8.1	Introduction.....	159
8.2	Hadoop Ecosystem and Core Modules	160
8.3	Hadoop Ecosystem Tools.....	162
8.3.1	Management and Monitoring Tools.....	162
8.3.2	Data Access and Retrieval	163
8.3.3	Data Processing.....	164
8.3.3.1	Data Processing Engines.....	164
8.3.4	Data Storage.....	166
8.3.4.1	Hadoop Distributed File System.....	166
8.3.4.2	Data Ingestion.....	166
8.3.4.3	DataBase Technologies	167
8.3.4.4	SQL-on-Hadoop	170
8.4	Applications of Hadoop Tools.....	170
8.4.1	Biology and Medicine	171
8.4.1.1	Parallel Genome Indexing with MapReduce	171
8.4.2	Material Science Research	172
8.4.2.1	Automated, High-Throughput Exploration of PSP Relationship Using MapReduce	173
8.5	Future Projects and Directions	174
	Acknowledgments.....	174
	References.....	175

8.1 Introduction

The amount of data being generated is enormous. Collecting information from companies regarding their operations that is accessible through millions of IoTs (Internet of Things) such as smartphones, automobiles, and wearables for multimedia and social network activities generates all sorts of data. The ability to mine intelligence from these data (more generally, “Big Data”) has become highly crucial for economic and scientific gains.

Researchers and IT practitioners are posed with new challenges in extracting meaningful insights because of the tri-dimensional expansion, volume, variety, and velocity (3Vs) (Laney, 2001), of data (Refer [Figure 8.1](#)). In addition to the more important 3Vs, there are two other Vs of Big Data, namely, variability and veracity. Storage and processing (analytics, etc.) of such data have very different requirements (batch, streaming/online) than that of traditional computing. This has led to the discovery of new paradigms in computing.

**FIGURE 8.1**

The more important 3Vs (velocity, volume, and variety) of the 5Vs of data.

Apache Hadoop (2016), a distributed large-scale data processing and storage framework with its wide variety of continuously evolving Ecosystem, has enabled the growth of Big Data requirements. Hadoop and its rich Ecosystem have become the de facto framework for Big Data analytics.

Due to its simplicity and acceptance for Big Data applications, Hadoop has a large community of active open-source and commercial support. This drives innovation in Hadoop, thereby adding more dimensionalities in the development of the Hadoop Ecosystem. The Hadoop Ecosystem has made data-intensive computing simple, allowing individuals with limited programming knowledge to extract information from data. This has reduced programming over-head.

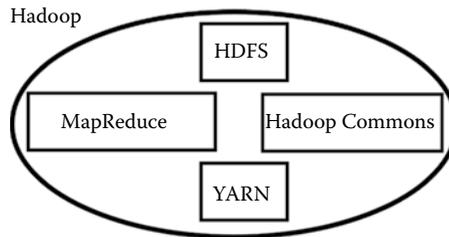
Due to the availability of a large variety of Hadoop tools, finding the most applicable tool for specific type of application is not easy. Our objective in this chapter is to provide readers the guidance in selecting appropriate Hadoop tools. We also emphasize how different data-intensive problems, irrespective of their field, can be modeled to Hadoop framework.

The chapter is organized as follows. Section 8.2 provides a brief overview of the Hadoop architecture and its foundations in providing support to build tools to cater to a wide variety of applications and services. In Section 8.3, we cover the breadth of the Hadoop Ecosystem tools applied across the life cycle of data (i.e., from storage to monitoring of data). We also emphasize on Hadoop Ecosystem for NoSQL databases. In Section 8.4, we discuss with examples the benefits of formulating large-scale data processing problems from state-of-the-art algorithms from different fields of research into existing Hadoop tools and frameworks (such as MapReduce and HDFS).

8.2 Hadoop Ecosystem and Core Modules

Hadoop, due to its simplicity, flexibility, and scalability (Landset et al., 2015), has evolved with a wide variety of projects for supporting Big Data application. Hadoop Ecosystem is the conglomeration of all such projects, tools, and techniques. Hadoop Ecosystem is built over Hadoop's core modules (namely, HDFS, YARN, MapReduce, and Commons).

Initially, Hadoop had only two core modules: its distributed file system, named Hadoop Distributed File System (HDFS) (Borthakur, 2008), and high-performance parallel data processing engine, based on MapReduce (Dean and Ghemawat, 2008). However, Hadoop has evolved over time to support complex data center workflows as well as other Big Data workloads. Now, it has the following four modules (Apache Hadoop, 2016) (as shown in [Figure 8.2](#)):

**FIGURE 8.2**

Hadoop and its core modules.

1. *Hadoop Distributed File System (HDFS)* is a distributed block-structured file system that stores data in large file system blocks (generally of 64/128 MB), known as chunks. HDFS follows master–slave architecture, where the master node (or name node) has the metadata information and each of these chunks is stored in different slave nodes (also known as data nodes). Chunks (and metadata) are tri-replicated across the cluster for fault tolerance and achieving higher availability (Borthakur, 2008). This model helps in preventing the data against disk failures as well as for faster processing of data for enabling parallel processing (moving computation to data), hence reducing unnecessary network traffic.
2. *Hadoop MapReduce* is the data processing framework based on Dean and Ghemawat (2008), which suggested a programming model with the same name. MapReduce implementation is used for processing of large data sets on a cluster with a distributed and parallel algorithm. It consists majorly of Map and Reduce phases. In the Map phase, the input data are organized into key–value pairs, and the sorted output is processed based on the key in the Reduce phase.
3. *Hadoop YARN* (“*Yet Another Resource Negotiator*”) (Kumar Vavilapalli et al., 2013) is the job scheduling and resource management (cluster management) module for Hadoop. Before YARN was added to the Hadoop core, Hadoop MapReduce had to manage both the infrastructure and parallel processing of data. The development of YARN was very important as YARN can run applications that do not follow MapReduce data processing framework. This increased Hadoop’s flexibility (see Section 8.3.3) to support other batch and stream processing frameworks (Storm [Hortonworks Apache Storm, 2016], Spark [Apache Spark, 2016], Flink [Apache Flink, 2016], etc.). The current MapReduce implementation is a YARN-based distributed data processing engine.
4. *Hadoop Commons* are common utilities (shared libraries of Java codecs, I/O, error detection, and security utilities [Landset et al., 2015]) which are needed to support other Hadoop modules. Hadoop Commons also provide abstractions to File System and OS level libraries required to initiate Hadoop.

Hadoop and its core modules due to their flexibility (open source, ability to encapsulate new features, and most importantly YARN support) have formed a rich ecosystem of tools and algorithms. These tools provide support for various solutions for different type of applications. With affordable and ease of access to high-performance computing (HPC) and cloud services, new avenues of industrial and academic research like bioinformatics, sentimental analysis, etc. are being fueled. The choice of tools and appropriate algorithms to suite an application is complex. This is due to varying requirements (real-time analytics,

data visualization, etc.), variety in type (also format), and a large volume of data to be processed.

In the next sections, we provide detailed information regarding the tools and algorithms that constitute the Hadoop Ecosystem. We also explain briefly the role of Hadoop Ecosystem in enabling Big Data and related research fields.

8.3 Hadoop Ecosystem Tools

Hadoop Ecosystem tools are software packages built on top of core Hadoop modules (as discussed above) to facilitate and provide support for various types of different applications (batch processing, real-time analytics, etc.).

Based on its applicability and functionality, the Hadoop Ecosystem is broadly classified into four categories:

1. Cluster management and monitoring
2. Data access and retrieval
3. Data processing
4. Data storage

Figure 8.3 shows the classification of Hadoop Ecosystem tools including names of some of the popular tools.

We now briefly discuss each of these categories and their functionality in the ecosystem along with the working of few of the most useful tools in each category (refer [The Ecosystem Table, 2016] for an exhaustive list of Hadoop projects).

8.3.1 Management and Monitoring Tools

This category consists of tools that can be applied for user interface, system deployment, monitoring, coordination, and organization for high-level operations required in Hadoop clusters. We discuss some of the popular and useful projects (refer Table 8.1).

Apache Ambari (Apache Ambari, 2016) project is a powerful tool for system administrators for installing Hadoop services (other data storage and processing tools) from scratch

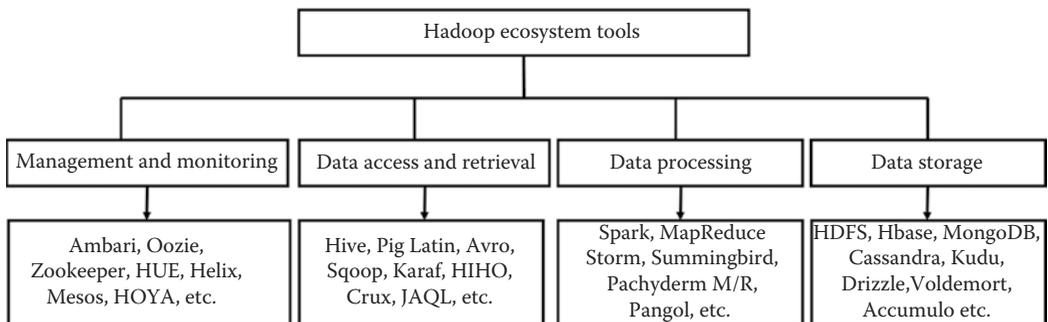


FIGURE 8.3
Classification of Hadoop Ecosystem tools.

TABLE 8.1

Cluster Management and Monitoring Tool Categorization

Functionality	Ambari	Zookeeper	HUE	Oozie
Installation and Setup	✓			
Configuration		✓		
User Interface	✓	✓	✓	✓
Workflow Scheduler				✓

across nodes. It is an easy-to-use Hadoop tool for monitoring, administrating, and life cycle management via web User Interface (UI) RESTful APIs.

Apache Zookeeper (Apache Zookeeper, 2016) developed by Yahoo! Research is a centralized coordination, configuration, and synchronization service to enable availability of distributed services. It has APIs for Java and C, and is responsible for simplification of the development process of building distributed systems, thereby providing agility and robustness. Famous projects like Apache HBase (Apache HBase, 2016), Storm (Hortonworks Apache Storm, 2016), and Kafka (Apache Kafka, 2016) use Zookeeper for managing the cluster (The Ecosystem Table, 2016).

Apache HUE (HUE, 2016) (Hadoop User Environment) is a web-based UI for Hadoop projects and is used to support Hadoop Ecosystem user operations. It enables user to view browsers for HBase (Apache HBase, 2016) and HDFS (Borthakur, 2008) operations, and job browsers for YARN (MapReduce), as well as provides interaction with Hive (Apache Hive, 2016), Sqoop (Apache Sqoop, 2016), Zookeeper (Apache Zookeeper, 2016), Impala (Cloudera Impala, 2016), Oozie (Apache Oozie Workflow Scheduler for Hadoop, 2016), Pig (Apache Pig, 2016), and other ecosystem tools.

Apache Oozie (Apache Oozie Workflow Scheduler for Hadoop, 2016) is a workflow scheduler to manage data processing jobs for MapReduce, Pig, Hive, etc. It also manages multiple jobs and different tools used at the same time by coordinating the sequence of tasks between tools and jobs. To schedule interval-based jobs, Oozie uses Directed Acyclic Graphs (DAG) using a trigger mechanism. Oozie is a very useful tool for utilizing multiple ecosystem tools deployed in complex distributed systems.

Other interesting projects include Apache Mesos, Myriad, Hortonworks HOYA, Apache Helix, etc. A brief overview of such tools can be found in The Ecosystem Table (2016).

8.3.2 Data Access and Retrieval

Data Access and Retrieval tools provide easy, fast, and reliable access to data. We focus our attention toward some of the popular data access and querying support technologies and data warehousing tools such as Hive (Apache Hive, 2016), Pig Latin (Apache Pig, 2016), and JAQL (IBM Hadoop Dev Jaql, 2016). These tools work on top of Hadoop YARN (Table 8.2).

TABLE 8.2

Data Access and Retrieval Tool Categorization

Functionality	Hive	Pig	JAQL	Crux
Data Warehousing	✓			
Data Set Analysis	✓	✓		
Document Format Querying			✓	
Database-Specific Reporting			✓	✓

Apache Hive (Apache Hive, 2016) is a popular data warehousing software system which is used for access and analysis of large data sets in distributed Hadoop storage (HDFS and NoSQL Databases as HBase, etc.) using SQL-type query language, known as *HiveQL*. It also supports projection of data in custom MapReduce when expressing the logic in HiveQL is difficult. Hive is best suited for traditional data warehousing jobs (ETL [Extract, Transform, and Load], reporting, data analytics, etc.) and is designed to maximize scalability and fault tolerance.

Apache Pig (Apache Pig, 2016) is a powerful large data set analysis tool which uses substantial parallelization for data processing. It consists of high-level textual language called Pig Latin, which provides ease in parallel execution for complex jobs. Pig Latin's compiler produces a sequence of Hadoop MapReduce jobs for data processing. This gives the scalability and ability to handle large data sets. Due to ease of programming, ability to handle large data sets as well as flexibility of developing own functions for reading, writing, and processing of data, Pig has found large acceptance in Big Data Analytics in both the industry and academia.

JAQL (IBM Hadoop Dev Jaql, 2016) is a popular query language for large-scale structured, semistructured, and unstructured data analysis tool (especially for JSON [JavaScript Object Notation] format). JSON is one of the most popular formats in which semistructured data are stored in the industry for machine learning (e.g., Social Network user activity information). JAQL uses Hadoop MapReduce for achieving parallelism and allows scalable SQL-like functionalities (select, join, filter, and group) of data in HDFS and NoSQL databases. JAQL is used in IBM working with its Hadoop platform, Infosphere BigInsights, to exchange data between storage, compute, and data analysis jobs (The Ecosystem Table, 2016).

Apache Crux (Apache Crux, 2016) is a project that focuses on being a reporting application specifically for column-based NoSQL database HBase. It uses HBase Java client for querying the data to produce web-based report (e.g., visualization in terms of graphs and plots of data).

8.3.3 Data Processing

Data processing engines have a huge impact on the overall performance of the system and their targeted application.

MapReduce has been the de facto processing engine for Hadoop for Big Data. MapReduce is well known for batch processing, but with increasing real-time analytics (Online Ads, recommendation systems [Netflix], surveillance, etc.), there is a greater need for the ability of stream processing. Moreover, due to high velocity of data, the processing speed of MapReduce is becoming a limitation due to its disk-based processing paradigm. This has led to the development of various other disk processing engines too, and in this section, we present some of them.

8.3.3.1 Data Processing Engines

YARN was developed to remove the MapReduce limitations and allows different applications to run which do not follow MapReduce. YARN has provided the flexibility to a large number of projects (Storm [Hortonworks Apache Storm, 2016], Spark [Apache Spark, 2016], Flink [Apache Flink, 2016], Hama [Apache Hama, 2016], etc.) to build new processing engines. Despite different architectures being proposed, due to its simplicity and large data processing capabilities, MapReduce is still the most widely used data processing framework. [Table 8.3](#) summarizes the different popular data processing engines.

TABLE 8.3

Data Processing Engines

Data Processing Engine	Processing Type	Data Fetched for Processing From	Latency
MapReduce	Batch	Disk	High
Spark	Batch and Stream	Memory and Disk	Low
Storm	Stream	Memory	Low

Apache MapReduce (Dean and Ghemawat, 2008), as discussed earlier, is a batch processing framework, which first reads the input data from the disk, and the chunks (HDFS) are processed in the Map phase. The Map phase produces outputs which have key–value pairs and are sorted in temporary files by an intermediate step known as shuffle, and these are the input to the Reduce phase. The Reduce phase then processes the data as per key and writes the final output to the disk (HDFS). MapReduce focuses on job throughput rather than latency sensitive. Complete understanding of MapReduce can be found in the Google’s introductory paper (Dean and Ghemawat, 2008).

Apache Storm (Hortonworks Apache Storm, 2016) is a YARN implementation of Hadoop for distributed real-time computation for a large volume of high-velocity data. Initially, it was developed for social media streams and is shown to be beneficial for machine learning, cyber security, and threat detection analysis applications. Storm architecture consists of spouts and bolts, which are represented by user-defined structure known as Topologies (which are directed acyclic graphs). The input data streams or sources are known as “spouts,” and the “bolts” does most of the computation and processing in the form of tuples from spouts or other bolts (Landset et al., 2015). Similar to MapReduce, Storm (Hortonworks Apache Storm, 2016) also follows a master–slave architecture, where the master is known as Nimbus.

Apache Spark (Apache Spark, 2016) is a popular data processing framework, which runs on top of Hadoop (it is a YARN implementation), and is used for streaming/online applications. Spark uses Resilient Distributed Datasets (RDD), which store data in-memory for computation rather than only disk I/O as in MapReduce (Landset et al., 2015). The in-memory caching of intermediate results as well as data reduces the I/O to disk and thereby gives Spark 10 times performance gain than MapReduce. Spark uses more RAM than disk and network I/O as compared to MapReduce. Spark needs high-end dedicated machines for productive results.

In Table 8.3, MapReduce is shown to be slower, but it covers a large variety of applications than either of Spark and Storm combined. Spark and Storm outperform MapReduce in real-time streaming applications/latency-specific applications (click stream analysis, twitter trends, etc.), but their consumption of a lot of memory and CPU speed is a major deciding factor for their success, while MapReduce works well also in commodity machines. MapReduce also takes care of those applications that need complete analysis of historical data (e.g., biomedicine, GIS, weather) and focuses on job throughput.

Efforts have been made to integrate MapReduce and Storm to achieve common batch and real-time processing simultaneously. *Twitter Summingbird* (Boykin et al., 2014) is a hybrid system that integrates batch processing (*MapReduce*) and stream/online processing (*Storm*), while keeping the same programming model. Summingbird is designed for analytics in Twitter and has paved the way for mitigating the tradeoffs of Batch and Stream processing together.

8.3.4 Data Storage

The overall performance of the system depends heavily on the storage (media, type, format, access, and retrieval and processing). With Big Data, storage has always been one of the major concerns. Due to low cost and high volume of data, Hard Disk Drives (HDDs) are the main storage media. There are on-going work on utilizing Hybrid Storage (HDDs + Solid State Drives [SSDs]) (Krish et al., 2014) (Iliadis et al., 2015) for Hadoop Data Centers. Though the storage technologies have evolved from increasing the areal density to the paramagnetic limit (e.g., Shingled Magnetic Recording Disks [Le Moal et al., 2012]), still access and retrieval of the data play a major role as the main contention is the movement of disk arm. Research directions to improve the storage have been on to decrease disk I/O time.

The sources, type, and format of data have changed. Earlier, mostly log data were analyzed, but now, they are nested, structured, unstructured, and semistructured data. The 5Vs (*velocity, volume, variety, variability, and veracity*) have given rise to various storage paradigms and tools which facilitate and aim to improve the performance.

Hadoop Storage Layer is broken into the following categories (refer [Figure 8.4](#)):

1. HDFS
2. Data Ingestion Tools
3. DataBase Technologies
4. SQL-on-Hadoop

8.3.4.1 Hadoop Distributed File System

HDFS has been the default file system for Hadoop that provides scalability and fault tolerance. HDFS assumes a write-once, read-many type of application scenario. For any change in the data, the entire file needs to be rewritten. It also works efficiently for large historical data processing which does not require random access. Moreover, with the variety of data (images, tweets, video, etc.) and application needs, there is a requirement of special tools for performance gains and different storage solutions. Most of the tools for distributed storage solutions are built to run on top of HDFS.

8.3.4.2 Data Ingestion

With the variety and large volume of different types of data, there has been a need for different tools for collecting, aggregating, and transferring data from various sources. A few of them are described in [Table 8.4](#).

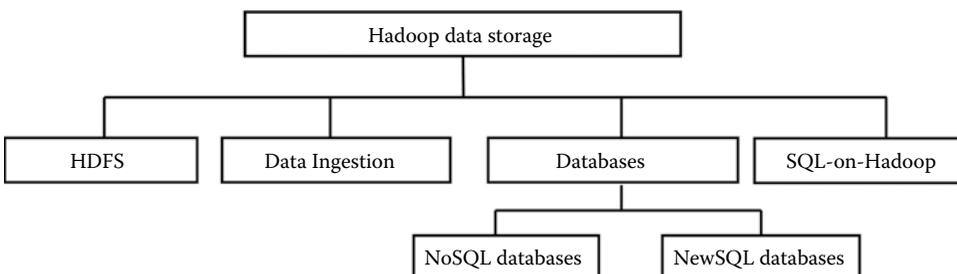


FIGURE 8.4
Classification of Hadoop Storage Layer.

TABLE 8.4

Data Ingestion Tools Categorization

Functionality	Flume	Chukwa	Sqoop	Kafka	Storm
RDBMS to HDFS	✓	✓	✓	✓	✓
HDFS to RDBMS			✓		
Stream Handling				✓	✓
Log Aggregator	✓	✓			

Apache Flume (Apache Flume, 2016) is an efficient distributed system for collecting, aggregating, and transferring large amounts of data to HDFS. Flume is simple, tunable, reliable, and based on streaming data flows. The data model allows for applications with real-time analytics.

Apache Chukwa (Apache Chukwa, 2016) is a large-scale distributed data collection and monitoring system built on top of the HDFS and MapReduce framework. It is a log aggregator and deals with incremental appending of logs across multiple machines, bridging incremental log handling and MapReduce. Chukwa and Flume have similar features, but Flume adopts a “hop-by-hop” model (Landset et al., 2015) by maintaining a centralized list of ongoing data flows stored in Zookeeper (Apache Zookeeper, 2016), while in Chukwa, the agents in each machine decide what data to send.

Apache Sqoop (Apache Sqoop, 2016) is used for bulk transfer of data between HDFS and other structured databases such as RDBMS. It functions quite the same as Flume but has an added functionality of transferring data from HDFS to RDBMS, which is very useful in archiving purposes. It offers two-way replication with incremental updates as well as snapshots.

Apache Kafka (Apache Kafka, 2016) is a scalable distributed publish–subscribe messaging system, which provides high-throughput processing of streaming data along with parallel loads with Hadoop. The messages in Kafka are persisted, thereby allowing clients to rewind streams as well as for offline analysis, bulk data can be imported to HDFS quickly and efficiently.

Apache Storm (Hortonworks Apache Storm, 2016) discussed in the Data Processing section also lies in data ingestion for streaming applications. Both Storm and Kafka work on streaming data, but Kafka is based on a publish–subscribe messaging system, while Storm is mostly about transferring a stream of messages into new streams (Landset et al., 2015).

Examples of other interesting projects include Netflix Suro (Netflix Suro, 2016), HIHO (Hadoop In Hadoop Out [HIHO], 2016), Facebook Scribe (Facebook Scibe, 2016), etc. being used for Data Ingestion in Hadoop systems.

8.3.4.3 DataBase Technologies

For efficient storage, the most vital decision is how, what, and where the data are stored. Traditional RDBMS have been the default database. Their architecture (row–column structure) poses negative impact on the overall performance for a large volume and velocity of data requirements by Hadoop-like applications as well as conventional RDBMS are unable to support the diaspora of a variety of data types (Moniruzzaman and Hossain, 2013). Apart from performance, the databases for such applications should be reliable and have proper integration with Hadoop Ecosystem tools.

The inefficiency of RDBMS in scalability and flexibility of data structures has given rise to alternative database management systems from NoSQL (NoSQL Databases: An

Overview, 2016) to the latest paradigms in the development of NewSQL (Moniruzzaman and Hossain, 2013) (NewSQL Overview, 2016).

We now provide details of some development in various database architectures and tools to support and enhance performance.

8.3.4.3.1 NoSQL (Not Only SQL)

Nonrelational databases, NoSQL (Not only SQL), support semistructured as well as unstructured data, which have been discussed in brief in the previous chapters, and so here we would focus on the database technologies that enable Hadoop storage requirements.

Out of the large NoSQL ecosystem, Apache HBase (Apache HBase, 2016), Cassandra (Apache Cassandra, 2016), and Accumulo (Apache Accumulo, 2016) are most aligned to Hadoop and by default use HDFS for storage. Based on the data models, the data type, and targeted applications, NoSQL databases are categorized as shown in Figure 8.5, and the Hadoop tools that support them are discussed subsequently.

8.3.4.3.1.1 Key–Value Databases Key–value stores are implemented on large hash tables and are the easiest of NoSQL databases. The client can insert or delete a key/value from the data store. The value is a data blob, which has a unique key, and the database stores it regardless of its structure. Due to hash table implementation, key–value stores are highly efficient, fast, and scalable distributed databases. Some of the popular data stores in this data model are LinkedIn Voldemort (Project Voldemort, 2016), MemCached (Issa and Figueira, 2012), Redis (Hortonworks Redis, 2016), etc.

LinkedIn Voldemort (Project Voldemort, 2016) (named after the famous Harry Potter villain, *Lord Voldemort*) is a distributed, persistent, fault-tolerant hash table based on key–value data store, used in LinkedIn for various operations. Voldemort is horizontally scalable for reads and writes with an API to decide data replication and placement for a large range of distributed Hadoop data centers. It also combines in-memory caching without requiring separate caching and is easily pluggable with existing Hadoop tools like *Avro* and *Thrift*. The project is still under construction. The details of other interesting in-memory data stores like *Memcached* with Hadoop can be found in Issa and Figueira (2012).

8.3.4.3.1.2 Document Databases These distributed databases retrieve and store documents in different formats such as JSON and XML. Primarily, these are nested key–value stores, with each key pointing to a collection of key–value stores (Landset et al., 2015). The documents (key–value stores) are in hierarchical tree data structures and are collection of similar types. The most popular of such databases are *MongoDB* (mongoDB FOR GIANT IDEAS, 2016), *CouchDB* (Apache CouchDB relax, 2016), *RethinkDB* (RethinkDB, 2016), etc., which also come with rich query language tools. *JAQL* (IBM Hadoop Dev Jaql, 2016) type

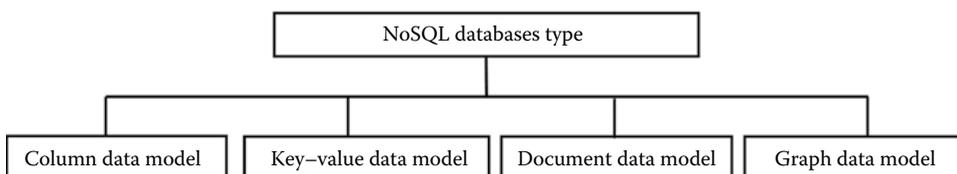


FIGURE 8.5

NoSQL Database Data Models.

Hadoop tools (as discussed in Section 8.3.2) are highly efficient for such document databases. All these databases store documents mostly in the JSON format and are highly scalable and use *MapReduce* for their processing of SQL-type operations (*JAQL*) and indexing. *MongoDB* (mongoDB FOR GIANT IDEAS, 2016) is used mostly in text processing and machine learning web and social engineering applications.

8.3.4.3.1.3 Column-Oriented Databases Column Data Model Databases have been one of the most popular and used NoSQL database solutions. This is because they store data in column families as rows (many columns are associated with a row key) rather than a row-column format as in RDBMS. Column families are groups of related data, which are mostly accessed together. A Super column consists the map of all columns (key name and value map of columns). Most of these data stores run on top of HDFS and are used for backing for Hadoop MapReduce outputs. Use of column-based databases has shown enormous performance gains for Hadoop workloads due to their alignment with Hadoop MapReduce-type characteristics.

Apache HBase (Apache HBase, 2016) is a very popular distributed column-based database and is used in Facebook's Messaging framework. It sits on top of HDFS and offers good random read and write IO performance. SQL-type queries are supported by HiveQL (Apache Hive, 2016) and other Hadoop querying tools. HBase tables are used for backing the outputs of Hadoop MapReduce jobs, and HDFS provides fault-tolerant storage. The main advantage of HBase is for environments where fast queries are required. *Cassandra* (Apache Cassandra, 2016) is a highly scalable distributed column-oriented clustered database, which does sharding in key ranges for ease of management and retrieval. It can run with or without HDFS, but with Hadoop, it has proved to be faster than HBase, which is attributed to their architecture being very much aligned to Hadoop MapReduce and Hadoop SQL-type querying tool (HiveQL, etc.) which use MapReduce.

Accumulo (Apache Accumulo, 2016) is a very robust and high-performance distributed solution built on top of Apache Hadoop, Zookeeper, and Thrift and is based on the Google's Big Table architecture. Other interesting projects include *Apache Kudu* (used by Chinese search engine, Baidu) and *Hypertable* (integrates with Spark and MapReduce and has fast SQL reads supported by Apache Drill) (The Ecosystem Table, 2016).

8.3.4.3.1.4 Graph-Based Databases Graph-based NoSQL databases are efficient for applications which have network analysis, social network connections, pattern detection, etc. The data are stored as graphs, with entities as nodes and relationships as edges, which have properties. The directions of the edges have significance in organizing relationships for finding valuable patterns. Various combinations of patterns and relationships can be found efficiently. Graphs-based model is more flexible than other database models.

TitanDB (TITAN Distributed Graph Database, 2016) is a large OLTP highly scalable and optimized distributed graph-based database for large clusters and capable of handling billions of nodes and edges. The analytics engine uses Hadoop MapReduce for processing graphs. Other important projects are Neo4j and ArrangoDB.

HBase (Apache HBase, 2016), Accumulo (Apache Accumulo, 2016), MongoDB (mongoDB FOR GIANT IDEAS, 2016), Voldemort (Project Voldemort, 2016), and Cassandra (Apache Cassandra, 2016) are few projects which have made NoSQL databases a very popular choice in Hadoop Data Centers. Due to their flexibility and scalability along with their integration in the Hadoop Ecosystem, Hadoop and NoSQL have provided a diaspora of highly efficient storage solutions for applications.

8.3.4.3.2 NewSQL

NewSQL (Moniruzzaman and Hossain, 2013) (NewSQL Overview, 2016) is relatively a newer and modern version of RDBMS that aims to provide scalability of NoSQL system for OLTP operations, while maintaining ACID guarantees (which NoSQL does not guarantee). The development of such a system is due to financial institutions order-processing need and to scale for high performance but are unable to use NoSQL, due to their strong dependency on ACID guarantees. *SensiDB* (SensiDB, 2016) is a high-performance NewSQL database that has fast key–value lookup and SQL-like query language with Hadoop integration.

8.3.4.4 SQL-on-Hadoop

The tools discussed in this section facilitate users in analyzing the data using any of the analytics engines and use simple instructions for fetching intelligence, rather than having to depend on programming it completely. They are fast, efficient, and in the background, they run MapReduce and other Hadoop tools to get the query or analytics.

The first SQL-on-Hadoop tool is *Hive* (Apache Hive, 2016), developed by Facebook, which query with its SQL-like language *HiveQL*. *Apache HCatalog* (Apache HCatalog, 2016) gives the users the relational view of data in HDFS. *Cloudera Impala* (Cloudera Impala, 2016) is a highly scalable and parallel technology for Hadoop, which provides low-latency SQL queries to HDFS and HBase (faster than Hive). *Facebook Presto* (Facebook Presto, 2016) is an open-source distributed SQL query engine for analytics over data stored in Hadoop and is faster than most of the SQL-on-Hadoop tools. Presto is used by companies like Facebook, Airbnb, and Dropbox for analytics over Hadoop (Facebook Presto, 2016).

SQL-on-Hadoop has revolutionized data analytics by removing the burden of writing code from scratch, do not rely on the ability of the coder, and has made analysis faster. Hence, Hadoop is now not restricted to the fraternity of people with good coding skills and has become more flexible to larger audience.

8.4 Applications of Hadoop Tools

Owing to the large Hadoop Ecosystem and NoSQL-type databases, there has been a large industrial and academic research focus on developing algorithms and solutions for extracting meaningful information from the data. The focus has not only been on Social Network Analysis, Machine Learning applications (click stream analysis, Content-based Advertisements, Tweet Sentimental Analysis, etc.), and Computer Sciences, but there have been a lot of research in the fields of Biomedicine, Materials, Meteorology, Agriculture, Environmental Research, etc. as well (refer [Figure 8.6](#)). Due to the simplicity and availability of tools, the applicability of fitting different models to the Hadoop framework is endless, and so Hadoop is now used by most of the companies and research laboratories in different fields of research.

In this section, we introduce problems from various fields of research which use Hadoop tools (and frameworks) and show how application of Hadoop has accelerated new discoveries and insights from data. With these examples, we also want to train readers on how they can mold their ideas dealing with large volumes of data (irrespective of their field of research/interest) into the Hadoop framework with the use of proper ecosystem tools and techniques without worrying much about the data volume.

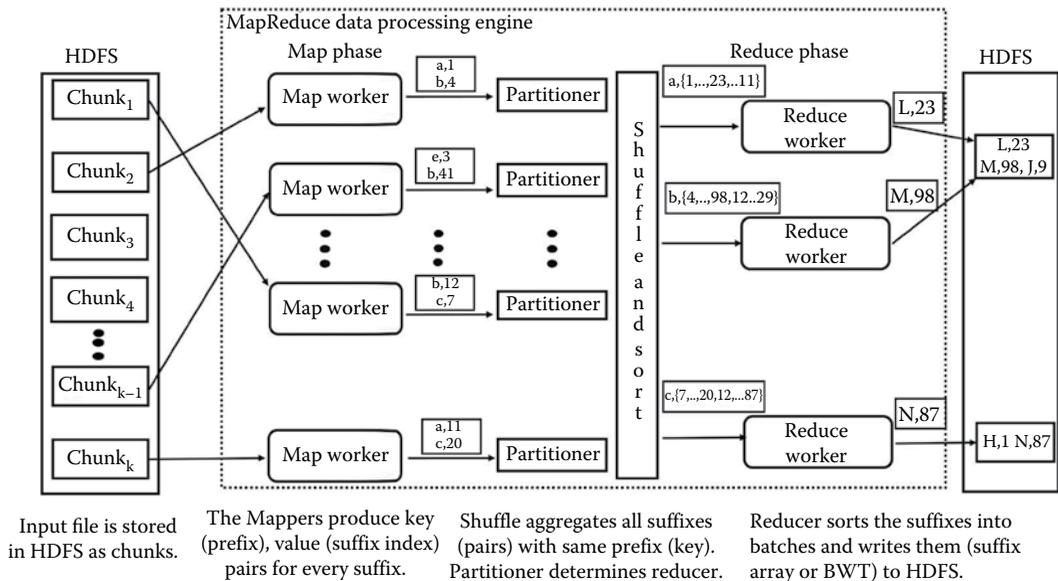


FIGURE 8.7

Working of MapReduce for BWT/SA construction.

these pairs (using same prefix). The reducers sort the suffixes in the batches using any sorted algorithm of strings and generate the SA/BWT. They improve on this construction by sampling partitioner to load balance number of suffixes per batch. For better results, they also use optimized recursive bucket sort to precompute keys and develop solution to detect biological ambiguities like repetitions on the fly.

By modeling a complex problem such as sequencing alignment into a parallel computing framework, researchers have achieved a near-linear performance. The scale of data and the amount of processing required for mining information earlier had inhibited computational biologists from analyzing such large data sets. This is a good example of the effectiveness of Hadoop MapReduce frameworks.

Other interesting algorithms used in computational biology and medicinal research are Hadoop-GIS (Wang et al., 2011), which implements a high-performance query system using Hadoop MapReduce and Hive for analysis of medical imaging, and GATK (O'Driscoll et al., 2013), which is a next-generation re-sequencing gene analysis tool kit.

O'Driscoll (O'Driscoll et al., 2013) lists an exhaustive list of the state-of-the-art tools and algorithms for enabling bioinformatics applications.

8.4.2 Material Science Research

Material Science is the design and discovery of new materials for engineering and industrial applications. All materials are made up of microstructures, which affect the performance of any device made from a material (Wodo et al., 2015). There is a lot of information contained in microstructures, and there are large sets of microstructure for various combinations. Obtaining optimal properties from these sets of microstructure can lead to new and efficient discoveries in material science.

This application offered the same challenge as the bioinformatics application. Process-Structure-Property (PSP) relationship using MapReduce (Wodo et al., 2015) has emphasized

the simplicity of MapReduce and used it to formulate their problem of finding microstructure traits into a MapReduce problem. This facilitated use of Hadoop MapReduce to automatically generate, explore, and identify highly correlated microstructure–property traits with minimal knowledge of HPC, and thus helped in designing of next-generation organic photovoltaics.

8.4.2.1 Automated, High-Throughput Exploration of PSP Relationship Using MapReduce

Microstructures contain information regarding a material's property, which themselves depend on the process by which a material is fabricated. The property of material can be tuned to show certain characteristics by choosing the correct processing parameters that result in a class of microstructures. The information regarding the tuned class of microstructures can be found by PSP relationship. Finding PSP relationships is highly nontrivial, with large combinatorics process and system variable searches and statistical identification, validation, quantification, and indexing of microstructures. Additionally, it has large and diverse data sets which are formed.

The authors have utilized the fact that large-scale exploration and characterization of combinatorics of complex processing operations of microstructures would result in data about the final microstructures. Moreover, in many such combinations, the result would be similar microstructures. Therefore, clustering of these similar microstructures together and focusing on the resultant structure would be highly beneficial. This observation has enabled the authors in formulating the PSP problem into MapReduce. Figure 8.8 shows the data flow for PSP using MapReduce.

The Process-to-Structure modeling is done for every sample point to compute the microstructures (using any of microstructure evolution framework). The input is a file with a list of processing conditions. In the Map phase, (Descriptor, Microstructures) pairs are produced which represent particular processing conditions. Here, the descriptors (generated using a graph-based microstructure descriptor [key] framework) serve as “Key,” and the microstructure produced is the “Value.” In the shuffle phase, clustering microstructures (values) with similar descriptors are clustered. These clusters define the same class of microstructure. Therefore, from these “representative microstructures,” the desired property can be modeled by using Structure-to-Property model (using a Microstructure Interrogation Framework). In the Reduce phase, the microstructures with same descriptors emit the properties.

The researchers (Wodo et al., 2015) have provided a way to harness the capabilities of Hadoop, by reformulating their problem of large-scale exploration of PSP relationships to the workflow of MapReduce. They have also mentioned that fast and simple solutions can

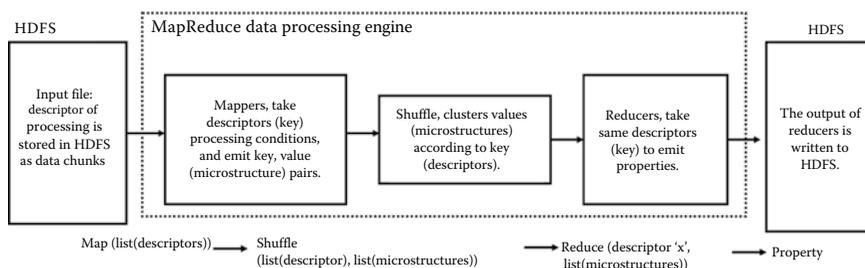


FIGURE 8.8

Working of PSP using MapReduce.

be found for other complex Material Science problems requiring large data processing by modeling them according to Hadoop MapReduce.

8.5 Future Projects and Directions

In the 2016 Hadoop Summit in San Jose (2016 Hadoop Summit, San Jose, California, USA, 2016), a full track is devoted to showcase new innovations in core Hadoop and other technologies for improving and creating next-generation Hadoop Ecosystem. As per Hortonworks (Mark Herring, 2016), this would include “services” as a new set of use cases for YARN development. The simplification of existing and new services for YARN would attract more interactive projects in the Hadoop Ecosystem. Furthermore, HDFS is being developed to not just support distributed file system but also other storage services. To add another dimensionality to HDFS, there are plans to use LSM trees (log-structured merge trees) (O’Neil et al., 1996) to support NoSQL databases like HBase (Apache HBase, 2016) in a better way.

The Hive community are working toward adding HBase as an alternative to store Hive’s metadata to make metadata access faster and reduce query planning time (Mark Herring, 2016). There is also development in incubation to improve data ingestion support of Hive (Apache Hive, 2016) for streaming applications using Apache Flume (Apache Flume, 2016) and Storm (Hortonworks Apache Storm, 2016).

Machine Learning and Data Mining tools and techniques have grabbed a lot of attention due to their applicability to a large number of industries (Landset et al., 2015). These include Entertainment (Netflix, Hulu, etc.), e-Commerce (Amazon, Flipkart, Lenskart, etc.), Social Network Ad (Facebook, Twitter, Google, etc.), and many more for gaining insight from data for business intelligence. *Apache Mahout* (Apache Mahout, 2016) is the most widely used Machine Learning tool, which is a simple distributed programming framework for building scalable algorithms. Its library contains many algorithms used for data mining and machine learning applications. Landset et al. (Landset et al., 2015) covers a plethora of machine learning tools, libraries, and algorithms. It also provides guidance for selection criteria as per application requirement.

The most useful resources to stay updated regarding developments in the Hadoop Ecosystem are through the Internet; research publications; attending relevant conferences (Hadoop Summit [2016 Hadoop Summit, San Jose, California, USA, 2016], etc.); technical web blogs of Big Data companies like Hortonworks (Mark Herring, 2016), Cloudera (Cloudera Impala, 2016), Facebook (Facebook Presto, 2016), etc.; and Hadoop meetup groups (Meetup App [Meetup App, 2016] is great way in finding people who have similar interests).

Acknowledgments

This research is funded in part by the Jerry R. Junkins Endowment and Philip and Virginia Sproul Professorship endowment at Iowa State University. The research implementation reported in this paper is partially supported by the HPC@ISU equipment at Iowa State

University, some of which has been purchased through funding provided by NSF under MRI grant number CNS 1229081 and CRI grant number 1205413. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

- 2016 Hadoop Summit, San Jose, California, USA. 2016. Retrieved June 9, 2016, from <http://hadoop-summit.org/san-jose/tracks/>
- Apache Accumulo. 2016. Retrieved June 9, 2016, from <https://accumulo.apache.org/>
- Apache Ambari. 2016. Retrieved June 9, 2016, from <http://ambari.apache.org/>
- Apache Cassandra. 2016. Retrieved June 9, 2016, from <http://cassandra.apache.org/>
- Apache Chukwa. 2016. Retrieved June 9, 2016, from <https://chukwa.apache.org/>
- Apache CouchDB relax. 2016. Retrieved June 9, 2016, from <http://couchdb.apache.org/>
- Apache Crux. 2016. Retrieved June 9, 2016, from <https://github.com/sonalgoyal/crux>
- Apache Flink. 2016. Retrieved June 9, 2016, from <https://flink.apache.org/>
- Apache Flume. 2016. Retrieved June 9, 2016, from <https://flume.apache.org/>
- Apache Hadoop. 2016. Retrieved June 9, 2016, from <http://hadoop.apache.org/>
- Apache Hama. 2016. Retrieved June 9, 2016, from <https://hama.apache.org/>
- Apache HBase. 2016. Retrieved June 9, 2016, from <https://hbase.apache.org/>
- Apache HCatalog. 2016. Retrieved June 9, 2016, from <https://cwiki.apache.org/confluence/display/Hive/HCatalog>
- Apache Hive. 2016. Retrieved June 9, 2016, from <https://hive.apache.org/>
- Apache Kafka. 2016. Retrieved June 9, 2016, from <http://kafka.apache.org/documentation.html>
- Apache Mahout. 2016. Retrieved June 9, 2016, from <http://mahout.apache.org/>
- Apache Oozie Workflow Scheduler for Hadoop. 2016. Retrieved June 9, 2016, from <http://oozie.apache.org/>
- Apache Pig. 2016. Retrieved June 9, 2016, from <https://pig.apache.org/>
- Apache Spark. 2016. Retrieved June 9, 2016, from <http://spark.apache.org/>
- Apache Sqoop. 2016. Retrieved June 9, 2016, from <http://sqoop.apache.org/>
- Apache Zookeeper. 2016. Retrieved June 9, 2016, from <http://zookeeper.apache.org/>
- Borthakur, D. 2008. HDFS Architecture Guide.
- Boykin, O., Ritchie, S., O'connell, I., and Lin, J. 2014. Summingbird: A framework for integrating batch and online MapReduce computations. *Proceedings of the VLDB Endowment* 7, 1441–1451.
- Cloudera Impala. 2016. Retrieved June 9, 2016, from <https://www.cloudera.com/products/apache-hadoop/impala.html>
- Dean, J. and Ghemawat, S. 2008. MapReduce: Simplified data processing on large clusters. *Communications of ACM*, 51(1), 107–113. <http://dx.doi.org/10.1145/1327452.1327492>
- Facebook Presto. 2016. Retrieved June 9, 2016, from <https://prestodb.io/>
- Facebook Scibe. 2016. Retrieved June 9, 2016, from <https://github.com/facebookarchive/scribe>
- Hadoop In Hadoop Out (HIHO). 2016. Retrieved June 9, 2016, from <https://github.com/sonalgoyal/hiho>
- Hortonworks Apache Storm. 2016. Retrieved June 9, 2016, from <http://hortonworks.com/apache/storm/>
- Hortonworks Redis. 2016. Retrieved June 9, 2016, from <http://hortonworks.com/partner/redis-labs/>
- HUE. 2016. Retrieved June 9, 2016, from <http://gethue.com/>
- Huo, H., Lin, S., Yu, Q., Zhang, Y., and Stojkovic, V. 2012. A MapReduce-based algorithm for motif search. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (pp. 2052–2060). IEEE. <http://doi.org/10.1109/IPDPSW.2012.255>

- IBM Hadoop Dev Jaql. 2016. Retrieved June 9, 2016, from <https://developer.ibm.com/hadoop/docs/biginsights-value-add/jaql/>
- Iliadis, I., Jelitto, J., Kim, Y., Sarafijanovic, S., and Venkatesan, V. 2015. ExaPlan: Queueing-based data placement and provisioning for large tiered storage systems. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2015 IEEE 23rd International Symposium on, 218–227. <http://dl.acm.org/citation.cfm?id=2865927>
- Issa, J. and Figueira, S. 2012. Hadoop and memcached: Performance and power characterization and analysis. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1), 10. <http://doi.org/10.1186/2192-113X-1-10>
- Kumar Vavilapalli, V., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., et al. 2013. Apache hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (P. 5)*. ACM, 13, 1–3. <http://doi.org/10.1145/2523616.2523633>
- Krish, K. R., Anwar, A., and Butt, A. R. 2014. hatS: A heterogeneity-aware tiered storage for Hadoop. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 502–511). IEEE. <http://doi.org/10.1109/CCGrid.2014.51>
- Landset, S., Khoshgoftaar, T. M., Richter, A. N., and Hasanin, T. 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1). <http://doi.org/10.1186/s40537-015-0032-1>
- Laney, D. 2001. Application Delivery Strategies. <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf>.
- Le Moal, D., Bandic, Z., and Guyot, C. 2012. Shingled file system host-side management of Shingled Magnetic Recording disks. In *2012 IEEE International Conference on Consumer Electronics (ICCE)* (pp. 425–426). IEEE. <http://doi.org/10.1109/ICCE.2012.6161799>
- Mark H. 2016. The Future of Apache Hadoop. Hortonworks. Obtained from <http://hortonworks.com/blog/future-apache-hadoop/>
- Meetup App. 2016. Retrieved June 9, 2016, from <http://www.meetup.com/>
- Menon, R. K., Bhat, G. P., and Schatz, M. C. 2011. Rapid parallel genome indexing with MapReduce. In *Proceedings of the Second International Workshop on MapReduce and Its Applications*. ACM, 51/58. Retrieved from <http://code.google.com/p/genome-indexing/>
- mongoDB FOR GIANT IDEAS. 2016. Retrieved June 9, 2016, from <https://www.mongodb.com/>
- Moniruzzaman, A. B. M. and Hossain, S. A. 2013. NoSQL Database: New era of databases for big data analytics—classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6(4). Retrieved from <http://arxiv.org/abs/1307.0191>
- Netflix Suro. 2016. Retrieved June 9, 2016, from <https://github.com/Netflix/suro>
- NewSQL Overview. 2016. Retrieved June 9, 2016, from <http://newsqldb.sourceforge.net/>
- NoSQL Databases: An Overview. 2016. Retrieved June 9, 2016, from <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>
- O’Driscoll, A., Daugelaite, J., and Sleator, R. D. 2013. “Big data,” Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5), 774–781. <http://doi.org/10.1016/j.jbi.2013.07.001>
- O’Neil, P., Cheng, E., Gawlick, D., and O’Neil, E. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4), 351–385. <http://doi.org/10.1007/s002360050048>
- Project Voldemort. 2016. Retrieved June 9, 2016, from <http://www.project-voldemort.com/voldemort/>
- RethinkDB. 2016. Retrieved June 9, 2016, from <https://www.rethinkdb.com/>
- SensiDB. 2016. Retrieved June 9, 2016, from <http://senseidb.github.io/sensei/>
- The Ecosystem Table. 2016. Retrieved June 9, 2016, from <https://hadoopecosystemtable.github.io/>
- TITAN Distributed Graph Database. 2016. Retrieved June 9, 2016, from <http://espeed.github.io/titandb/>
- Wang, F., Aji, A., Liu, Q., and Saltz, J. H. 2011. Hadoop-GIS: A High Performance Spatial Query System for Analytical Medical Imaging with MapReduce.
- Wodo, O., Zola, J., Pokuri, B. S. S., Du, P., and Ganapathysubramanian, B. 2015. Automated, high throughput exploration of process–structure–property relationships using the MapReduce paradigm. *Materials Discovery*. <http://doi.org/10.1016/j.md.2015.12.001>

9

Hadoop Ecosystem Tools and Algorithms

Kevin Berwind, Marco Xaver Bornschlegl, Michael Kaufmann, Felix Engel,
Michael Fuchs, Dominic Heutelbeck, and Matthias Hemmje

CONTENTS

9.1	Introduction	178
9.2	Introduction of the Hadoop Architecture, Modules, and Projects	179
9.2.1	Introduction to Information and Knowledge Generation and Management in the Age of Big Data	179
9.2.2	Apache Hadoop	179
9.2.2.1	HDFS: The Hadoop Distributed File System	180
9.2.2.2	Hadoop Map-Reduce	180
9.2.2.3	YARN	181
9.2.3	Apache Hadoop Projects	181
9.2.3.1	HBase	181
9.2.3.2	Hive	181
9.2.3.3	Spark's Machine Learning Library	181
9.2.3.4	Zookeeper	181
9.2.3.5	Oozie	182
9.2.3.6	Falcon	182
9.2.3.7	Flume	182
9.2.3.8	Sqoop 2	182
9.2.4	Apache Hadoop and AAI Support	182
9.3	Introduction in Data Collection, Management, and Curation	183
9.3.1	Data Integration and Collection Based on Hadoop	183
9.3.1.1	Sqoop 2	183
9.3.1.2	Flume	183
9.3.1.3	Kafka	184
9.3.2	Data Management and Curation Based on Hadoop	184
9.3.2.1	Falcon	184
9.3.2.2	Atlas	184
9.3.3	Archival and Preservation Based on Hadoop	185
9.4	Introduction into Data Analytics with Hadoop	186
9.4.1	Data Analysis and Indexing	186
9.4.1.1	Hive	186
9.4.1.2	Pig	186
9.4.1.3	Mahout	186
9.4.1.4	Spark's Machine Learning Library	186
9.4.2	Data Filtering/Querying and Reporting	187
9.4.3	Machine Learning Tools and Techniques	187

9.4.3.1	Apache Mahout	187
9.4.3.2	Apache Spark's Machine Learning Library	187
9.5	Data Visualization, Interaction, and Perception with Hadoop	188
9.5.1	Visualizations with Hadoop	188
9.5.2	External Tool and Component-Based Data Visualization	188
9.5.3	Interaction with Visual Data Views	189
9.5.4	Interaction with the Visualization Pipeline and Its Transformation Mappings	189
9.6	Ecosystem Cross-Domain and Cross-Organizational Processing Methodologies ..	190
9.6.1	Big Data Analysis Infrastructures Based on IVIS4BigData Reference Model	190
9.6.2	Big Data Analysis Process Model Based on CRISP4BigData Reference Model	192
9.6.3	Cloud-Based Hadoop Systems and Services	193
9.7	Introduction to Data Insight and Effectuation	194
9.7.1	Introduction to Knowledge-Based Decision and Support	194
9.7.2	Introduction to Data-Driven Information Services	195
9.7.3	Introduction to Data-Driven Knowledge Extraction Services	195
	References	195

9.1 Introduction

The volume of enterprise data in data centers increases between 35% and 50% each year (Beath et al. 2012). Through new wide-ranging developments in the area of information and sensor technology and the increasing speed of analysis and processing, data especially of calculation-intensive and analytics methods will be fostered. The collected mass data stem from the internal and external corporate divisions and consist of unstructured data such like text data, processing information, (database-) tables, graphics, videos (Beath et al. 2012), e-mails, feeds, and sensor data (Freiknecht 2014).

The acquisition of decisional relevant mass data is known as “Big Data.” At the beginning, the term was defined by companies which had to handle a large amount of fast-growing data. The information technology research and advisory company Gartner, Inc. defines the term as follows “Big data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making” (Gartner 2015). The American market research company IDC defines Big Data as follows: “Big Data technologies as a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis” (Carter 2011).

Today's companies have the pressure to deal with such Big Data because their business is fundamentally built on collecting and analyzing this large amount of data. There are several software frameworks and methods to handle Big Data like CUDA, Storm, Mahout, and Hadoop (Chao et al. 2015). In this context, Google developed the Google File System (Ghemawat et al. 2003) and Map-Reduce (Dean and Ghemawat 2010) to collect and process such data. Both technologies are the foundation for the Apache Hadoop and the Hadoop File System (EMC Education Services 2015), which are licensed under the Apache Software Foundation (Apache Software Foundation 2015a,b). The Apache Hadoop Ecosystem is

based on internal modules like HDFS, YARN, Map-Reduce (Singh and Reddy 2014), and several projects like Falcoon, Atlas, Hive, Pig, MLlib, and Flume (Landset et al. 2015).

This chapter outlines deliberated software frameworks based on Hadoop in a cloud-based ecosystem to support small- and medium-sized businesses to get access to Big Data Analytics. In addition, the chapter outlines some reference models and architectures to operate a Hadoop Ecosystem.

9.2 Introduction of the Hadoop Architecture, Modules, and Projects

9.2.1 Introduction to Information and Knowledge Generation and Management in the Age of Big Data

The collection of data and information is a trend that runs like a common thread through the information and knowledge management. Even companies collect, as already said, internal and external data for their data warehouses. As a consequence, not only the need for optimal and adequate storage increases, but also new solutions for a next level of data security, access management, data governance, archiving, and protection of value added (Lehner 2014). The uncontrolled increase of data and information leads not per se into a better knowledge support. Besides, the information and knowledge management is especially a well deployment of knowledge system responsible for a sustainable and holistically as a solution.

The definition of value of data, consequential information and knowledge is methodically not that simple. The value of information could be done over several ways. The first one is a subjective definition of value, whereby users get interviewed about the value of the used information. The second one is the usage of an overserved value, whereby the result of the decision process will be compared with and without the used information (Lehner 2014). The value of information depends even on the use case, to which the information is used.

In an economic use case, it is a goal to minimize the costs. This problem is regarded of the so-called Information Lifecycle Management (ILM). The ILM values the information automatically, to optimize the use of resources and the accessibility. A usual opportunity to define the value of information is to rate them with the usage over time (frequency of usage, utilization time, source of usage). Even more frequent or longer accesses is done, the better is the value of the information (Lehner 2014). The following subchapters will describe how information technology such as software of the Apache Hadoop Ecosystem could support the collection, management, processing, and archiving of data and how companies use the data and extract information and knowledge to get an advantage toward their competitors. Furthermore, some methods will be offered to deal with the transformation from raw data into visualizations and to manage the whole data processing initiating by the Business Understanding to Data Archiving.

9.2.2 Apache Hadoop

Within the Apache Hadoop project, licensed under the Apache Software Foundation (Apache Software Foundation 2015a,b), software is developed for scalable and distributed computing (Apache Hadoop Project 2016). The Hadoop software library is a Java framework used for the distributed storing of data and the parallel processing of large data

sets across clusters (Freiknecht 2014). Hadoop operates in a horizontal scale-up cluster to run single servers as well as thousands of machines (Apache Hadoop Project 2016). It is designed to run on commodity hardware which is comparatively keen and easy to purchase. Furthermore, the Apache architecture is designed to deliver high availability with fault detection to handle failures at the application layer (Apache Hadoop Project 2016).

The Hadoop project includes various modules as introduced in the following (Apache Hadoop Project 2016):

- Hadoop Distributed File System (HDFS): A distributed file system that provides storage and processing power to process data across the cluster.
- Hadoop YARN: A programming model that provides distributed processing of applications across the cluster and managing of resources for those.
- Hadoop Map-Reduce: A programming framework for parallel processing of large data sets according to the two-phase processing with mapper and reducer.

9.2.2.1 HDFS: The Hadoop Distributed File System

The HDFS, which is inspired by the Google File System, is a distributed storage used by Hadoop applications. A HDFS cluster consists of a NameNode that manages the system metadata, data systems, and directory structures and DataNodes that store the actual data (Freiknecht 2014).

The Hadoop's HDFS is supporting the following features:

- Distributed storage and distributed processing using commodity hardware.
- Scalable, fault tolerant, and extremely simple to expand.
- HDFS is highly configurable with a default configuration, which is well suited for many installations.
- Hadoop is written in Java, thus it is supported on all major platforms.
- Hadoop supports shell commands to access and interact with HDFS.
- NameNode and DataNodes have built on web servers (check current status of the cluster).

The HDFS provides a set of functions that allow writing data into the data system and reading these data from the system. In case that the NameNode receives a file from the client, which should be saved within the file system, it needs two information, first the size of the data blocks and second the quantity of replicas, distributed over the cluster, to divide the file into several data blocks which have a size between 64 and 129 MB (Freiknecht 2014). The NameNode seeks after those DataNodes as many replicas postulated by the client. After the client gets the addresses of the data notes back, the client starts to write the data into one DataNode. The Node is going to replicate this data over all DataNodes.

9.2.2.2 Hadoop Map-Reduce

Map-Reduce is a software framework that processes high volumes of data in parallel on a Hadoop cluster. The frameworks are based on the following three phases to process data: Map phase, Combine phase, and Reduce phase. The Map phase divides the input data into

independent data units and builds an index with key–value pairs and assigns them to data units which are processed within the phase (Freiknecht 2014). Individual entries of the index could be existing redundant and will be aggregated within a Combine phase which generates a new key–value pair. Finally, the Reduce phase aggregates (add up) or thins out the values of the index to get the best value (Freiknecht 2014).

9.2.2.3 YARN

Apache YARN is a framework for job scheduling and cluster resource management. The new architecture introduced in Hadoop 0.23 divides the architecture of the Job Tracker functions into two new components: the resource management and job life cycle management (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). The Resource Manager and per-machine Node Manager manage the allocation of compute resources and user processes for all applications within the cluster. The Application Master manages the scheduling, coordinates the applications, and restarts the application on failure. An application corresponds to a single job or a DAG (directed acyclic graph) of jobs (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3 Apache Hadoop Projects

9.2.3.1 HBase

HBase is a distributed and scalable NoSQL database of the Apache Foundation based on Google’s Bigtable (Freiknecht 2014). HBase supports both linear and modular scaling, which means the servers are hosted on commodity class servers and could expand from 10 to 20 or more servers immediately to double storage and processing capacity (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3.2 Hive

Apache Hive offers data warehouse facilities for reading, writing, and managing datasets located in the distributed storage using SQL. It enables an easy access to the data to perform tasks such as ETL (extract, transform, and load), reporting, and analysis. Apache Hive grants access to the Apache HDFS or other data storage systems like Apache HBase (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3.3 Spark’s Machine Learning Library

Apache MLlib is a scalable machine learning library based on Apache Spark using Java, Scala, Python, and SparkR. MLlib and Spark produced high-quality algorithms, 100x faster than the known Map-Reduce. MLlib offers algorithms like clustering, classification, and linear regression (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3.4 Zookeeper

Hadoop is processed in a distributed environment with distributed storage and applications over several clusters. Zookeeper is a so-called Distributed Coordination Service (Freiknecht 2014), used as centralized service for maintaining configuration information, providing distributed synchronization, naming and supporting group services over the Hadoop cluster (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3.5 Oozie

Apache Oozie is an Apache-based project that provides a server-based workflow schedule system to manage Apache Hadoop jobs. Oozie steers and manages those jobs via Directed Acyclical Graphs (DAGs) of actions. The workflows could be started manual and triggered by time (frequency) and the availability of data. Apache Oozie is a scalable, reliable, and extensible system that integrates a Hadoop stack supporting a set of Hadoop jobs based on Map-Reduce, Hive, Pig, and Sqoop. It supports java code snippets and shell-based commands as well (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Oozie workflows are defined with HPDL (Hadoop Process Definition Language) which is a XML process definition language similar to JBOSS jBPM.

9.2.3.6 Falcon

Apache Falcon is a data governance, feed processing, and feed management framework that contains, for example, a data life cycle management mechanism, a data retention/archival management function, and the capability to define hot and cold storage tiers within a Apache Hadoop Cluster (Apache Software Foundation 2016a, b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.2.3.7 Flume

Apache Flume is a distributed, reliable, and available application for the efficient collecting, aggregating, and moving of large data sets. In the focus of Flume are data and data streams from all over the enterprise, such as data from internet of things or social media (Apache Software Foundation 2012a,b,c).

9.2.3.8 Sqoop 2

Apache Sqoop 2 is a tool for an efficient transfer of data between structured, semistructured, and unstructured data sources. Structured data, for example, are stored in relational databases, semistructured data are stored in Hbase, and unstructured data are stored in the HDFS (Apache Software Foundation 2013).

9.2.4 Apache Hadoop and AAI Support

In a deployment scenario, authentication and authorization are important aspects to be addressed. For example, it may manipulate or delete data within HDFS. Authentication covers the aspect of asserting the identity of a user or process, while authorization is the process of determining if the authenticated user may perform a requested action.

In HDFS, authorization is realized through so-called access control lists, and they behave similar to traditional UNIX file systems, taking into account users, owners of resources, and groups of users. The authentication and identification of the group's users can be realized directly through the host operating system or by configuring the Hadoop to integrate with LDAP, Active Directory, or Kerberos services.

In larger organizations, authentication and authorization are sometimes realized by more complex AAIs (Authentication and Authorization Infrastructures). The European Commission offers a "Study on Authentication and Authorisation Platforms for Scientific Resources in Europe" (European Commission 2012) which provides a comprehensive overview of the topic.

The need to access resources in different administrative domains in combination with the evolution of web technologies, collaborative and international research, and the increasing number of systems requiring authentication has imposed new requirements on access management technologies for education and research (European Commission 2012). The study describes further that a “Provisioning [of] user accounts for each application that users wish to access does not scale well in a highly distributed and collaborative environment that crosses multiple administrative domains and national boundaries.” The study evaluates the feasibility of delivering an AAI to support the emergence of data platforms for access and preservation of (scientific) information. Those AAIs offer often an accounting function to determine how much resources users consume to collect statistical information and to record authentication proceedings.

9.3 Introduction in Data Collection, Management, and Curation

9.3.1 Data Integration and Collection Based on Hadoop

The integration and collection of data within a Hadoop ecosystem could be done over several opportunities. The best-known Apache projects that handle Data integration and Collection are described following.

9.3.1.1 Sqoop 2

Apache Sqoop 2 is a tool for an efficient transfer of data between structured, semistructured, and unstructured data sources. Structured data, for example, are stored in relational databases, semistructured data are stored in Hbase, and unstructured data are stored in the HDFS (Apache Software Foundation 2013). Apache Sqoop 2 is divided into a client and a server application. The client communicates with the server over a REST API; based on that, API, a web-based graphical user interface, is added to expose a simple user interface. The Sqoop 2 server will be installed and configured on server-side that brings the benefit that all connectors will be managed and configured in one place (Apache Software Foundation 2012a,b,c).

9.3.1.2 Flume

Apache Flume is a distributed, reliable, and available application for efficient collecting, aggregating, and moving of large data sets. In the focus of Flume are data and data streams from all over the enterprise, such as data from internet of things or social media (Apache Software Foundation 2012a,b,c). The input and output data streams that transfer data from a source to a destination system and processed parallel are called “Runners.” All incoming data sets that are going to process from Apache Flume are called “Source.” Data sets processed from Flume are called “Sink.” There is a service called “Channel” between Source and Sink data. The Channel service is the processing and connection layer of Flume. Channels, Sources, and Sinks are operating in a Java Virtual Machine called “Agent,” whereby the Sources and Sinks are operating asynchronously with the Events staged in the Channels (Apache Software Foundation 2012a,b,c). There is another term called “Event” which consists of a header and a body. A Source consumes Events in a special format and is delivered by an external source like a web server. An Event could be stored in one or more Channels. The Channel, which is a passive store, holds the Event

until the Event is consumed by a Sink (Apache Software Foundation 2012a,b,c). One type of Channel in Flume is the so-called FileChannel which is using the local file system as store. A Sink is responsible for the deletion of an Event from a Channel (FileChannel) and to transfer it to an external destination system or repository like the HDFS.

9.3.1.3 Kafka

Apache Kafka is a distributed publish–subscribe messaging system that is designed to process messages fast, scalable, and durable. Kafka could handle hundreds of megabytes of reads and writes per second from hundreds or thousands of client systems (Apache Foundation 2016c). Kafka is designed to maintain messaging feeds in categories called topics, which are portioned and replicated across the cluster. The feeds of messages that are published by a system to the Kafka topic are called producers. Systems that subscribe to a topic to consume that feeds of messages are called consumers. Kafka is operating on a cluster comprising one or more servers where each server is called broker. The communication between the producers and consumer is done with a simple and high-performance TCP protocol (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). The Messages are represented simply as byte arrays stored in formats like String, JSON, and Avro. It is also possible to connect a key to each published message. In the case, the producer guarantees that all messages with the same key will arrive at the same portion (Holoman and Gwen 2014).

9.3.2 Data Management and Curation Based on Hadoop

The management and curation of data could be done by the support of the Apache projects, Faclon and Atlas, described in the following.

9.3.2.1 Falcon

Apache Falcon is a data governance tool that defines, schedules, and monitors the management of data policies. Furthermore, it allows administrators to define their data pipelines based on automatically deployed Apache Oozie workflows (Muisse 2014). Falcon supports the implementation of relationships between data and processing elements within a Hadoop cluster. Furthermore, it offers services for feed retention, replications, and archival. In addition to that, it manages the data lifecycle in a central place where an administrator can define and manage policies and pipelines for data collection, processing, and export (Hortonworks, Inc. 2016). Falcon provides also access to a metastore or catalog such as Hive or HCatalog and is able to tag data with business-related metadata (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). The governance tool manages “[the] Replication across on-premise and cloud-based storages [such as] [...] Microsoft Azure and Amazon S3” and defines hot or cold storage tiers within a Hadoop cluster (Hortonworks, Inc. 2016).

9.3.2.2 Atlas

Apache Atlas, which is an incubator project from the Apache Software foundation, is a Data Governance and Data (Lifecycle) Management tool developed among others from Aetna, Hortonworks, and Merck (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Apache Atlas has the task to handle all data within the ecosystem. Therefore,

Atlas classifies the Data for an Auditing and creates a Meta data store and considered security and policy requirements (Apache Software Foundation 2015a,b). Atlas adds Meta data based on libraries called “hooks” from various systems like Apache Hive, Apache Falcon, and Apache Sqoop. Therefore, Atlas uses Apache Kafka as a notification server for communication between Atlas itself and the named systems like Apache Hive.

Apache Atlas is, as already said, an incubator project with a lot of planed features. At the moment, the project offers the following features (Apache Software Foundation 2015a,b):

- *Data Classification*: Apache Atlas imports and defines taxonomy for business-oriented annotations for data. Furthermore, it defines, annotates, and captures relationships between data sets and underlying elements like source, target, derivation, and processes.
- *Centralized Auditing*: The application collects security access information for every other application, process, and interaction with data. It captures operational information for execution, steps, and activities.
- *Search and Lineage*: Atlas offers predefined navigation paths to explore data classifications and audit information, as well as text-based search features to locate relevant data across the ecosystem.
- *Security and Policy*: Based on data classification schemes, attributes, and roles, Apache Atlas rationalizes a compliance policy. It manages the advanced definitions of policies to prevent the derivation data based on classifications.

9.3.3 Archival and Preservation Based on Hadoop

The archiving or the classification of data could be done in the Hadoop Ecosystem in several manually or automatically processes. Apache Hadoop offers a solution called Archival Storage to separate growing storage capacity from compute capacity (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Apache Hadoop describes that “Nodes with higher density and less expensive storage with low compute power are becoming available and can be used as cold storage in the clusters” (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Based on policies, hot data can be moved to cold data storage. It is possible to add nodes to the cold storage anytime, so growing is possible without dependencies of the compute capacity in the cluster. Hadoop offers different storage policies to allow files to be stored in different storages according to the used policy. The following storage policies are available (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p):

- *Hot*: “for both storage and compute. The data that is popular and still being used for processing will stay in this policy. When a block is hot, all replicas are stored in DISK.”
- *Warm*: “partially hot and partially cold. When a block is warm, some of its replicas are stored in DISK and the remaining replicas are stored in ARCHIVE.”
- *Cold*: “only for storage with limited compute. The data that is no longer being used or data that needs to be archived is moved from hot storage to cold storage. When a block is cold, all replicas are stored in ARCHIVE.”
- *ALL_SSD*: “for storing all replicas in SSD.”

- *One_SSD*: “for storing one of the replicas in SSD. The remaining replicas are stored in DISK.”
 - *Lazy_Persist*: “for writing blocks with single replica in memory. The replica is first written in RAM_DISK and then it is lazily persisted in DISK.”
-

9.4 Introduction into Data Analytics with Hadoop

9.4.1 Data Analysis and Indexing

The Apache Software Foundation offers and manages several projects, which has the goal to query, filter, index, and analyze data via an SQL language, the high-level language Pig Latin, and several Machine Learning algorithms. The most noted projects are presented in the following.

9.4.1.1 Hive

Apache Hive was started in 2008 as incubator project under the Apache Software Foundation, and it offers data warehouse facilities like reading, writing, and managing of data sets located in a distributed storage. Hive provides an easy access to data over those data warehouse functions such as extraction, transformation, and loading (ETL) for a reporting and analysis. Hive could access Apache HDFS and Apache HBase via SQL.

9.4.1.2 Pig

Apache Pig is based on the high-level language Pig Latin. Goal of this project is analysis of a high volume of data. Similar to Apache Hive, Pig Latin will be translated in a Map-Reduce job and run on Hadoop. It seems that Apache Hive and Pig are very similar, but they should be used if the data must be extracted, transformed, and loaded (ETL) within a complex process. In contrast, Hive offers a simple querying and ad hoc analysis of the data via SQL language (Apache Software Foundation 2011).

9.4.1.3 Mahout

Apache Mahout is an extensible programming framework for building scalable and performant machine learning applications (Apache Software Foundation 2016a, b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Mahout could be operated over the Hadoop cluster, and it offers Data Mining functions such as clustering, classification, and recommendation based on Map-Reduce jobs (Freiknecht 2014).

9.4.1.4 Spark's Machine Learning Library

The Machine Learning Library of Spark is a scalable and distributed machine learning library that provides common algorithms including classification, regression, clustering, collaborative filtering, and dimensionality reduction (Apache Software Foundation 2016a, b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.4.2 Data Filtering/Querying and Reporting

As already said, Hive is a data warehousing infrastructure based on Apache Hadoop, which provides massive scale-out capabilities with a high degree of fault tolerance for distributed storages and processing based on commodity hardware (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Hive uses on the one hand Apache HDFS to store data and on the other hand Map-Reduce to translate queries to operate them within the Hadoop Ecosystem. Hive is designed to support a data summarization, querying, filtering, and analysis of large data sets. It provides an SQL language that enables querying, summarization, and analysis of data. Moreover, Hive offers users multiple capabilities to integration their own custom analysis such as User Defined Functions (UDF) (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). Hive is less designed for online transaction processing; it is especially designed and used for traditional data warehousing tasks. The architecture of Hive is built up on a central component called Driver, which handles the queries sent from various interfaces and accompanies them from over the whole life cycle from compiling, over the optimization and execution. A second component called Compiler translates queries, sent from the Driver, in a DAG that has one or more Map-Reduce jobs. The driver sends the Map-Reduce jobs to an Execution Engine called Resource Manager. Apache Hive uses a relational Database (named Metastore) to capture Meta Data about databases, schemas, tables, data types, and owner (Freiknecht 2014). Furthermore, it provides access over a Thrift-Client, a Command Line Interface (CLI), and the so-called Hive-Web-Interface (HWI). It is possible to use and configure Hue as an alternative user interface to interact with Apache Hive (Apache Software Foundation 2016j). Even Hive Provides a Connection over JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity) drivers (Apache Software Foundation 2015a,b).

9.4.3 Machine Learning Tools and Techniques

The Apache Software Foundation offers two Machine Learning projects which could be used to analyze a large volume of data within Apache Hadoop or the in-memory solution Apache Spark. The meant projects are Apache Mahout and Apache Spark's Machine Learning Library.

9.4.3.1 Apache Mahout

Apache Mahout is a Machine Learning Project licensed under the Apache Software foundation. Mahout could analyze data distributed over the Hadoop cluster. Mahout offers therefore a set of algorithms like clustering, classification, and recommendation. Mahout was first based on Map-Reduce jobs to execute several algorithms on data but is, for some time, past deprecated. However, Mahout (Apache Mahout 0.12.0) offers algorithms for Spark, H₂O, and Flink. A full list of algorithms is available online at <https://mahout.apache.org/> (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

9.4.3.2 Apache Spark's Machine Learning Library

The Machine Learning Library of Spark is a scalable and distributed machine learning library that provides common algorithms including classification, regression, clustering, collaborative filtering, and dimensionality reduction (Apache Software Foundation 2016 a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). The Machine Learning library is divided into two packages

called `spark.mllib` (contains the original API, built on Resilient Distributed Datasets [RDDs]) and `spark.ml` (provides higher-level Machine Learning Pipelining API). MLib could execute algorithms based on distributed datasets, representing data as RDDs, which are a fault-tolerant collection of elements that can be operated in parallel (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p). One important thing to note is that Spark's MLib only contains algorithms which could be processed parallel on the cluster. Some Machine Learning algorithms are not contained because they are not able to run on parallel systems. Therefore, MLib contains several research algorithms designed for clusters, such as distributed random forests, K-Means, and alternating least squares. In Spark 1.0 and 1.1, the interfaces of MLib operate on a relatively low level and provide various functions to run tasks. In some cases, it will be necessary to implement a workflow for a learning pipeline (e.g., splitting up the data into a training and test data set or tying several combinations of parameter input). Therefore, it could be used in the second package of Spark's Machine Learning Library called `spark.ml`, which provides a pipeline API to build workflows (or pipelines) (Karau et al. 2015).

9.5 Data Visualization, Interaction, and Perception with Hadoop

9.5.1 Visualizations with Hadoop

There are several capabilities to visualize Big Data or results of a Big Data Analysis in Hadoop with external Tools and Frameworks. The Apache Software Foundation is not offering a project that offers functions for data visualization. Only Cloudera develops an open-source analysis, visualization, collaboration, and workflow management portal licensed under Apache License (Version 2.0) called Hue. In case a whole portal is not needed, it is possible to program and implement visualizations with an external programming framework such as D3.js (D3 means Data-Driven Documents). D3 framework uses HTML, SVG, and CSS to develop dynamics diagrams based on JavaScript. The D3 community offers a lot of different graphics, diagrams, maps, and charts such as Bubble Charts, Bullet Charts, Chord Diagrams, Hierarchical Edge Bundling, and Tree maps (Bostock 2016). More examples could be found online at www.github.com/d3/d3/wiki/Gallery.

9.5.2 External Tool and Component-Based Data Visualization

There are many external software tools that could access the Hadoop Ecosystem over standardized interfaces such as JDBC or ODBC. The most noted tools are, for example, Microsoft Excel or Tableau Desktop from Tableau Software (2016). Microsoft Excel has advantage to perform ad hoc analysis with a set of pie charts, bar diagrams, and line charts. Furthermore, employees need no, in most cases, detailed trainings and instructions because they use Microsoft Excel mostly (Freiknecht 2014). Another named tool is Tableau Desktop which likewise an external data analyzing and visualization tool. Tableau offers the capability to analyze, explore, and visualize data. Especially in the area of data visualization, Tableau shows strengths concerning the variety of diagrams, charts, maps, and dashboards.

9.5.3 Interaction with Visual Data Views

Big Data analysis is based on different perspectives and intentions. To support management functions in their ability of making sustainable decisions, Big Data analysis specialists are filling the gap between Big Data analysis result consumers and Big Data technologies. Thus, these specialists need to understand their consumers/customers intentions as well as a strong technology watch, but are not the same like developers, because they care about having impact on the business (Upadhyay and Grant 2013). Deduced from this perspectives and intentions, there are different use cases and related user stereotypes that can be identified for performing Big Data analysis collaboratively within an organization. Users with the highest contextual level, for example managers of different hierarchy levels of such organizations, need to interact with visual analysis results for their decision-making processes. On the other hand, users with low contextual levels, like system owners or administrators, need to interact directly with data sources, data streams, or data tables for operating, customizing, or manipulating their systems. Nevertheless, user stereotypes with lower contextual levels are interested in visualization techniques as well in case these techniques are focusing on their lower contextual levels. Finally, there are user stereotype perspectives in the middle of those excesses, representing the connection between user stereotypes with low and high contextual levels. As a consequence, from these various perspectives and contextual levels, it is important to provide the different user stereotypes a context-aware system for their individual use cases. “The ‘right’ information, at the ‘right’ time, in the ‘right’ place, in the ‘right’ way to the ‘right’ person” (Fischer 2012). One could add to this citation with “the right competences” and/or with “the right user empowerment.”

9.5.4 Interaction with the Visualization Pipeline and Its Transformation Mappings

Information Visualization (IVIS) has emerged “from research in human-computer interaction, computer science, graphics, visual design, psychology, and business methods” (Thomas and Cook 2006). Nevertheless, IVIS can also be seen as a result of the question for interchanging ideas and information between human, keeping with Rainer Kuhlen (Kuhlen 2004), because of the missing direct way. The most precise and common definition of IVIS as “the use of computer-supported, interactive, visual representations of abstract data to amplify cognition” stems from Card et al. (1999). To simplify the discussion about information visualization systems and to compare and contrast them, Card et al. (1999) defined a reference model, which is illustrated in Figure 9.1, for mapping data to visual forms for human perception.

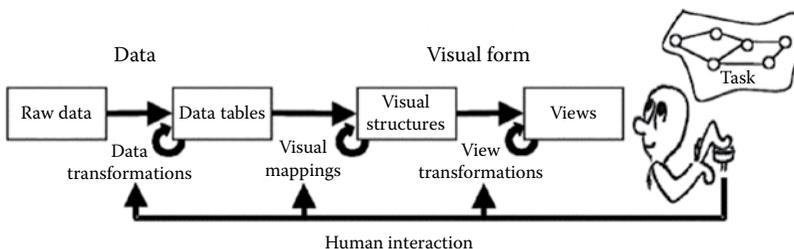


FIGURE 9.1
IVIS reference model.

In this model, arrows lead from Raw Data to visual data presentation of the raw data within a cognitive efficient IVIS based on a Visual Structure and it is rendering of a view that is easy to perceive and interact with for humans. The arrows in this model indicate a series of data transformations, where each arrow might indicate multiple chained transformations. Moreover, additional arrows from the human at the right into the transformations themselves indicate the adjustment of these transformations by user-operated controls supporting human-computer interaction (Kuhlen 2004). Data Transformations map raw data such as text data, processing information (database tables, e-mails, feeds, and sensor data) into data tables which define the data with relational descriptions and extended metadata (Beath et al. 2012; Freiknecht 2014). Visual Mappings transform data tables into visual structures that combine spatial substrates, marks, and graphical properties. Finally, View Transformations create views of the visual structures by specifying graphical parameters such as position, scaling, and clipping (Kuhlen 2004). “Although raw data can be visualized directly, data tables are an important step when the data are abstract, without a direct spatial component” (Kuhlen 2004). Therefore, Card et al. define the mapping of a data table to a visual structure, that is, a visual mapping, as the core of reference model; this operation translates the mathematical relations within data tables to graphical properties within visual structures.

9.6 Ecosystem Cross-Domain and Cross-Organizational Processing Methodologies

9.6.1 Big Data Analysis Infrastructures Based on IVIS4BigData Reference Model

The hybrid refined and extended IVIS4BigData reference model (Figure 9.2), an adaptation of the IVIS reference model, in combination with Kaufmann’s BDM reference model (Kaufmann 2016) is achieved to cover the new conditions of the present situation with advanced visual interface opportunities for perceiving, managing, and interpreting Big Data analysis results to support insight. Integrated into the underlying reference model for BDM, which illustrates different stages of BDM, the adaptation of the IVIS reference model represents the interactive part of the BDM life cycle.

According to Card et al., arrows that indicate a series of (multiple) data transformations lead from raw data to data presentation for humans. However, instead of collecting raw data from a single data source, multiple data sources can be connected, integrated by means of mediator architectures, and in this way globally managed in Data Collections inside the Data Collection, Management, and Curation layer. The first transformation, which is located in the Analytics layer of the underlying BDM model, maps the data from the connected data sources into Data Structures, which represent the first stage in the Interaction and Perception layer. The generic term Data Structures also includes the use of modern Big Data Storage Technologies (e.g., NoSQL, RDBMS, HDFS), instead of using only data tables with relational schemata. The following steps Visual Mappings, which transform data tables into Visual Structures, and View Transformations, which create Views of the Visual Structures, by specifying graphical parameters such as position, scaling, and clipping, do not differ from the original IVIS reference model. As a consequence, only interacting with analysis results leads not to “added value” for the optimization of, for example, research results or business objectives. Furthermore, no process steps are currently located within the Insight

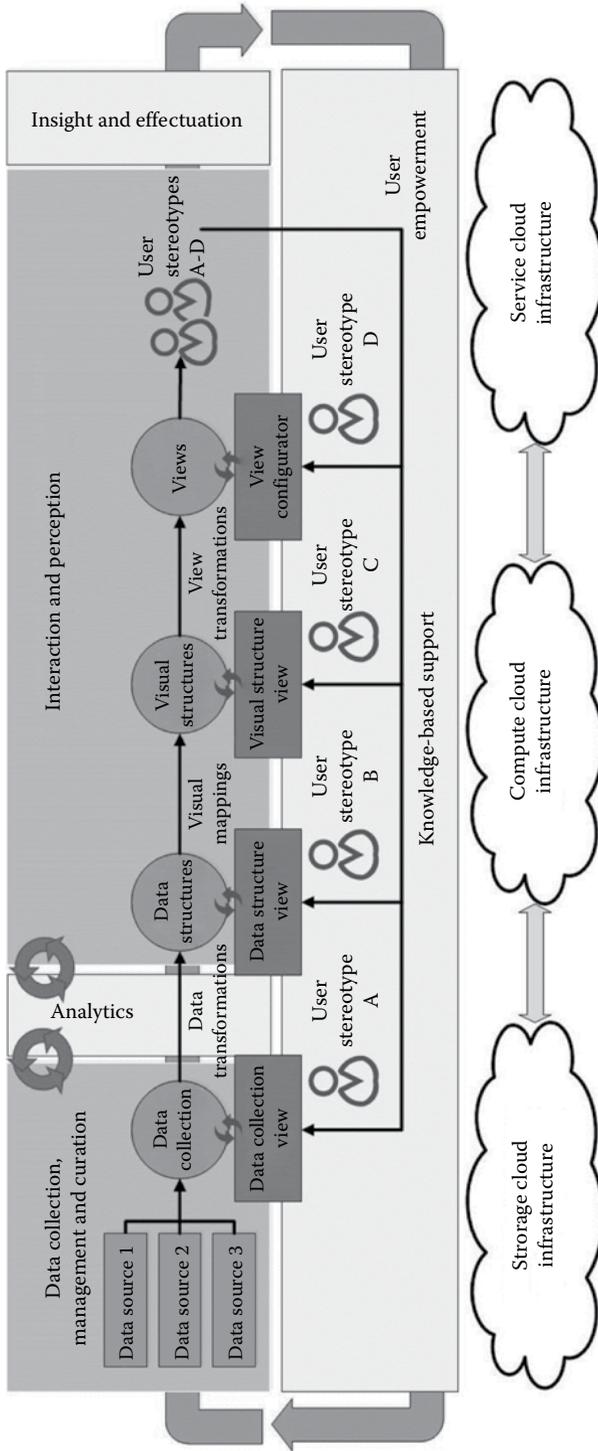


FIGURE 9.2 Data reference model. (Bornschlegl, M. X. et al. 2016. Towards a reference model for advanced visual interfaces supporting big data analysis in virtual research environments, *Proceedings on the International Conference on Internet Computing (ICOMP)*: 78-81. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).)

and Effectuation layer because such “added value” is rather generated from knowledge, which is a “function of a particular perspective” (Nonaka and Takeuchi 1995) and will be generated within this layer by combining the analysis results with existing knowledge.

The major adaptations are located between the cross-functional Knowledge-Based Support layer and the corresponding layers above. As a consequence, from the various perspectives and contextual levels of Big Data analysis and management user stereotypes, additional arrows lead from the human users on the right into multiple Views. These arrows are illustrating the interaction between user stereotypes with single process stages and the adjustments of the respective transformations by user-operated controls to provide “the ‘right’ information, at the ‘right’ time, in the ‘right’ place, in the ‘right’ way to the ‘right’ person” (Fischer 2012), within a context-aware and user-empowering system for individual use cases. Finally, the circulation around the layers clarifies that IVIS4BigData is not an one-time-process, because the results can be used as the input for a new process circulation.

9.6.2 Big Data Analysis Process Model Based on CRISP4BigData Reference Model

The CRISP4BigData reference model (Cross Industry Standard Process for Big Data) is based on Kaufmann’s Big Data Management reference model (Kaufmann 2016), Bornschlegl’s IVIS4BigData, and an enhancement of the classical Cross Industry Standard Process for Data Mining (CRISP-DM) developed by the CRISP-DM consortium, existing of DaimlerChrysler (later Daimler-Benz), SPSS (later ISL), NCR Systems Engineering Copenhagen, and OHRA Verzekeringen en Bank Groep B.V., with the target to handle the complexity of Data Mining projects (Chapman et al. 2000).

The CRISP4BigData reference (Figure 9.3) model is based on Kaufmann’s five phases of Big Data Management such as “Data Collection, Management, and Curation,” “Analytics,” “Interaction and Perception,” “Insight and Effectuation,” and “Knowledge-Based Support”. Within each of this phases are process elements implemented which handle special

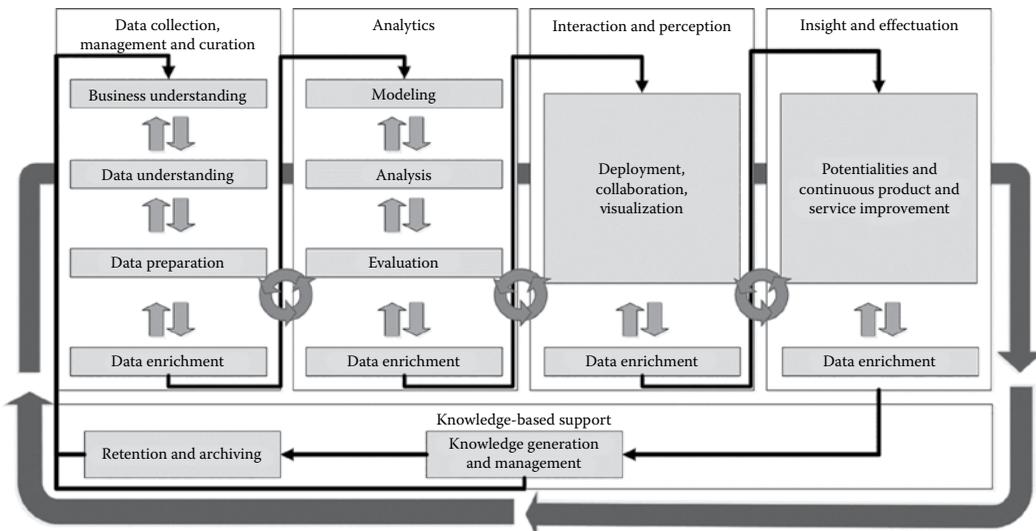


FIGURE 9.3 CRISP4BigData reference model. (Bornschlegl, M. X. et al. 2016. Towards a cross industry standard process to support big data applications in virtual research environments, *Proceedings on the Collaborative European Research Conference (CERC)*: 56–59. Cork, Ireland.)

instructions and tasks to deal with Big Data and Analysis processes and projects. Business Understanding describes the goals and problem statements, derived from a project plan to develop a targeted deployment of Big Data Analysis Methods. The process element Data Understanding deals with the collection of internal and external data, the description of data types and source systems. The Data Preparation manages the preparation of the Data over a transformation or Data Cleansing (or update, enlarge, and reduce) to increasing the data quality. All Data Enrichment process elements in the CRISP4BigData have the task to enrich the data base with useful Meta Data according to the whole process, to get insight into process information to increase the quality of the process, or to update the Knowledge-Based Support phase with information such as: Who uses which data or analysis methods? Who is a knowledge carrier? Where the data are from? Where is the data used?

The Modeling phase describes the development and implementation of statistical or mathematical model based on an adequate data model. The Analysis is based on the statistical or mathematical model and describes the deployment of a Big Data analysis method or algorithms to analyze the available data model. The Evaluation rates the result of the analysis and the process. The phase evaluates also the precision, usefulness, novelty, and significance of the result. The Deployment, Collaboration, and Visualization element deals with the deployment, visualization of the analysis result, and manages the distribution of the right result to the right persons (collaboration). The Potentialities and Continuous Product and Service Improvement element steers the management of potentialities and cares about a continuous improvement process to work out some new opportunities for the company. The process element Knowledge Generation and Management manages that the generated insights and knowledge do not get lost. The Retention and Archiving step manages a long-term retention, and archiving of the data, it manages also the classification of hot, cold, and warm data (tiers). Finally, the circulation around the whole phases shows that the CRISP4BigData reference model is not an on-time process. It is partially useful or needed to repeat the whole process to deal with new information obtained during the project or to improve the whole process.

9.6.3 Cloud-Based Hadoop Systems and Services

The term Cloud or Cloud Computing is originated a few years ago, in a time within which it became very popular to store and process data not local on a computer but on a server system within a (outsourced) datacenter (Veit et al. 2014). The advantage of Cloud Computing is to design hardware and software requirements dynamical according to how much data should be stored or processed. Cloud Computing could be defined with five characteristics. The first characteristics are the so-called on-demand self-service, which describes an independent interaction between users and the Cloud Computing services without an intervention of the service provider. The second characteristic is the network access which grants a real-time access to the Cloud Computing services over multiple devices like desktop computers, laptops, tablets, and smartphones. The third characteristic is resource pooling, which manages an adequate assignment of IT resources like processing power or memory space. The fourth characteristic Cloud Computing is the fast adaptability and elasticity of the service. That means least the resources could be adjusted automatically or manually according to customer's use case. The last one is the measurability of the service utilization, which grants an exact billing of the used accomplishments (Leimeister 2015). The access to those Cloud Computing services ensued most of the time over the internet. The access of the Cloud Computing services distinguishes per used access model. The models are called Private Cloud, Public Cloud, Hybrid Cloud, and

Community Cloud. The Private Cloud is solely used within an organization (e.g., within a company). The Public Cloud, in contrast, grants access for breadth publicity (e.g., online storage provider, Dropbox). A Hybrid Cloud is a mixture of private and public clouds, with a goal of high security (private cloud) and high scalability (public cloud). For example, companies could store critical data within the private cloud and uncritical data within the public cloud. The last one is the Community Cloud, which allows a common usage of several organizations (e.g., organization of the same branch, like research) (Leimeister 2015). There are some Hadoop Cloud Computing providers like Amazon, Microsoft, Google, and the European Grid Infrastructure, which offer private, public, and community cloud spaces. Google, for example, provides a fully managed cloud-based data warehouses, data flow processing (batch and stream), data analysis, or visualization services based on Apache Hadoop and Spark (Google, Inc. 2016). Amazon's cloud-based Elastic Map-Reduce (AmazonEMR) offers web services for Big Data processing, analysis, and warehousing based on Hadoop (Amazon Web Services, Inc. 2016). The European Grid Infrastructure offers a Community Cloud (cluster of Private Clouds), named Federated Cloud, which provides European researchers processing power for their operating systems, tools, and Hadoop or Spark clusters (European Grid Initiative 2016).

9.7 Introduction to Data Insight and Effectuation

9.7.1 Introduction to Knowledge-Based Decision and Support

The use of Decision Support Systems (DSS) shows the increasing management effectiveness and productivity in execution of decision problems (Zhongzhi 1987). Management decisions have a number of characteristics, "for example, (1) their high significance such as a lot of money are involved and usually a lot of people's interests are at stake; (2) time limitation—many business decisions have to be made within very tight time constraint in order for businesses to capture the market opportunity; (3) high degree of complexity—business decisions are often situated in rather complex environment [...] (4) high degree of uncertainty—business environment can change quickly, especially under the current unstable financial situation around the world, business decisions need to take account of this uncertainty in order to adapt to the evolving global market" (Liu et al. 2015). As a result, leading managers have searched for technology (or rather IT) support. Therefore, various types of DSS have been developed to support the managers to make a decision. The architecture of a typical DSS consists of three components: database management system (DBMS), model base management system (MBMS), and user interaction management system (or human-computer interface [HCI]). The architecture of the DSS fits for several years the requirements of decision makers (managers) and provides the right information, at the right time (Liu et al. 2015). But decision makers need not only a good information base; they need also a good knowledge base to make good judgments. The scenarios, decision makers have to deal with, are very complex, and it is hardly possible for them to have all the knowledge and expertise, needed for the decision-making. To solve this problem, a concept of a new generation of Knowledge-Based Decision Support Systems (KB-DSS) was designed in the 1980s (Liu et al. 2015). The new architecture design consists of some components known from the DSS: DBMS, MBMS, and HCI. In contrast, new is the knowledge base and inference engine which reference to the knowledge base and infer new knowledge (Liu et al. 2015).

9.7.2 Introduction to Data-Driven Information Services

The types of data and information within a company are various and originated during production process, financial transactions, social media transactions, and maintenance processes (Section 9.1). Though the interconnection between manufacturing technologies and classical information technologies such as Cloud Computing comes along with new opportunities, it increases the volume of data simultaneously. Even the implementation of new processors, storages, sensors, and transmitters within machines, products, and materials increases the volume of collected data (Siemens 2014). The goals of Data-Driven Information Services are to collect and analyze the data in real time and reckon an adequate solution and give a recommendation. It is important for the analysis process that the quality of data is high and the right data in the right time are available. Implementation of a cloud-based platform could be useful because of its scalability and low costs. Companies that need Data-Driven Service do not have to operate own servers or complete computer centers.

As already said, Google, for example, provides a fully managed cloud-based data warehouses, data flow processing (batch and stream), data analysis, or visualization services based on Apache Hadoop and Spark. Amazon's cloud-based Elastic Map-Reduce (AmazonEMR) offers web services for Big Data processing, analysis, and warehousing based on Hadoop. For instance, Siemens offers a Data-Driven Service called Energy Analytics, which should collect adequate data and analyze them to save cost for energy noticeably. A second emphasis is on the so-called Condition Monitor, which controls the current status of systems. It monitors the availability of machines and facilities and recognizes wastage of material or other issues to solve and delegate those during a predictive maintenance (Siemens 2014).

9.7.3 Introduction to Data-Driven Knowledge Extraction Services

Data-Driven Knowledge Extraction Services describe the generation of knowledge from structured (e.g., relational database) and unstructured (e.g., documents, text, media, images) data. The underlying knowledge base (structured and unstructured data) needs to be in a machine-readable and machine-interpretable format (Auer et al. 2012). A Knowledge Extraction Service is related to methods of the Information Extraction (e.g., Natural Language Processing, NLP) and Extract, Transform, Load process (ETL, data warehouse method), but in addition to it, *“goes beyond the creation of structured information or the transformation into a relational schema”* (Auer et al. 2012). Furthermore, it is required to reuse existing formal knowledge (reusing of identifiers or ontologies) or to develop a reusable schema based (e.g., Resource Description Framework, RDF) on the data sources (Auer et al. 2012). Apache Hadoop offers a lot of different projects which could support the analysis of structured and unstructured data. Hive, for example, offers complete data warehouse facilities (e.g., ETL), Apache Jena offers a set of libraries to start work with RDF data (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p), Apache UIMA provides a set of algorithms which are able to detect entities between persons, names, and places or to recognize sentence boundaries or languages (Apache Software Foundation 2016a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p).

References

Amazon Web Services, Inc. 2016. Amazon EMR mit Hadoop-Framework. Accessed June 05, 2016. <http://aws.amazon.com/de/elasticmapreduce/>

- Apache Hadoop Project. 2016. Welcome to Apache™ Hadoop®!. Available online at <http://hadoop.apache.org/>. Checked on August 11, 2016.
- Apache Software Foundation. 2011. Apache Pig Index. Accessed May 07, 2016. <http://cwiki.apache.org/confluence/display/PIG/Index>
- Apache Software Foundation. 2012a. Apache Flume: Flume 1.6.0 Developer Guide. Accessed May 10, 2016. <http://flume.apache.org/FlumeDeveloperGuide.html>
- Apache Software Foundation. 2012b. Apache Flume. Accessed May 11, 2016. <http://flume.apache.org/>
- Apache Software Foundation. 2012c. The Apache Software Foundation Blogging in Action: Apache Sqoop. Accessed May 11, 2016. http://blogs.apache.org/sqoop/entry/apache_sqoop_highlights_of_sqoop
- Apache Software Foundation. 2013. Apache Sqoop Documentation. Accessed May 09, 2016. <http://sqoop.apache.org/docs/1.99.6/index.html>
- Apache Software Foundation. 2015a. Apache Project List. Accessed December 02, 2015. <http://www.apache.org/index.html>
- Apache Software Foundation. 2015b. Data Governance and Metadata Framework for Hadoop. Accessed May 14, 2016. <http://atlas.incubator.apache.org/0.6.0-incubating/index.html>
- Apache Software Foundation. 2016a. About Hive's Functionality. Accessed May 05, 2016. <http://cwiki.apache.org/confluence/display/Hive/Home>
- Apache Software Foundation. 2016b. Apache Jena: A Free and Open Source Java Framework for Building Semantic Web and Linked Data Applications. Accessed June 09, 2016. <http://jena.apache.org/>
- Apache Software Foundation. 2016c. Apache Kafka: A High-Throughput Distributed Messaging System. Accessed May 11, 2016. <http://kafka.apache.org/documentation.html>
- Apache Software Foundation. 2016d. Apache UIMA: What is UIMA? Accessed June 09, 2016. <http://uima.apache.org/>
- Apache Software Foundation. 2016e. Archival Storage, SSD & Memory. Accessed June 01, 2016. <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/ArchivalStorage.html>
- Apache Software Foundation. 2016f. Apache ZooKeeper. Accessed May 10, 2016. <http://zookeeper.apache.org/>
- Apache Software Foundation. 2016g. Falcon: Feed Management and Data Processing Platform. Accessed May 10, 2016. <http://falcon.apache.org/index.html>
- Apache Software Foundation. 2016h. Hive Client. <http://cwiki.apache.org/confluence/display/Hive/Tutorial>
- Apache Software Foundation. 2016i. Hive Tutorial. Accessed May 08, 2016. <http://cwiki.apache.org/confluence/display/Hive/Tutorial>
- Apache Software Foundation. 2016j. Hive Tutorial. <http://gethue.com/>
- Apache Software Foundation. 2016k. Machine Learning Library (MLlib) Guide. Accessed May 6, 2016. <http://spark.apache.org/docs/latest/mllib-guide.html>
- Apache Software Foundation. 2016l. MapReduce NextGen aka YARN aka MRv2. Accessed June 10, 2016. <http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/index.html>
- Apache Software Foundation. 2016m. The Team. Accessed May 13, 2016. <http://atlas.incubator.apache.org/team-list.html>
- Apache Software Foundation. 2016n. Spark MLlib. Accessed May 14, 2016. <http://spark.apache.org/mllib/>
- Apache Software Foundation. 2016o. Accessed May 05, 2016. <http://spark.apache.org/>
- Apache Software Foundation. 2016p. What is Apache Mahout? Accessed May 14, 2016. <http://mahout.apache.org/>
- Auer, S., Hellmann, S., Stadler, C., and Unbehauen, J. 2012. Knowledge Extraction from Structured Sources. Search Computing—Broadening Web Search. Berlin, Heidelberg: Springer. <http://dx.doi.org/10.1007/978-3-642-34213-4>

- Beath, C., Becerra-Fernandez, I., Ross, J., and Short, J. 2012. Finding Value in the Information Explosion. *MIT Sloan Management Review*. <http://sloanreview.mit.edu/article/finding-value-in-the-information-explosion/>
- Bornschlegl, M. X., Berwind, K., Kaufmann, M., and Hemmje M. 2016. Towards a reference model for advanced visual interfaces supporting big data analysis in virtual research environments, *Proceedings on the International Conference on Internet Computing (ICOMP)*: 78–81. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
- Bornschlegl, M. X., Berwind, K., Kaufmann, M., and Hemmje M. 2016. Towards a cross industry standard process to support big data applications in virtual research environments, *Proceedings on the Collaborative European Research Conference (CERC)*: 56–59. Cork, Ireland.
- Bostock, M. 2016. Data-Driven Documents. Accessed May 05, 2016. <http://d3js.org/>
- Card, S. K., Mackinlay, J. D., and Shneiderman, B. 1999. *Readings in Information Visualization: Using Vision to Think*. The Morgan Kaufmann Series in Interactive Technologies. San Francisco, CA: Morgan Kaufmann. (This article was published in *Readings in information visualization: Using vision to think* by Card, S. K., Mackinlay, J. D., Shneiderman, B., Copyright Elsevier 1999.)
- Carter, P. 2011. *Future Architectures, Skills and Roadmaps for the CIO*. Edited by IDC. Available online at <http://www.sas.com/resources/asset/BigDataAnalytics-FutureArchitectures-Skills-RoadmapsfortheCIO.pdf>, updated on 2011.
- Chao, H., Lai, C., Tsai, C., and Vasilakos, A. 2015. Big data analytics: A survey. *Journal of Big Data*. <http://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0030-3>
- Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. 2000. CRISP-DM 1.0: Step-by-Step Data Mining Guide. Accessed June 09, 2016. <ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf>.
- Dean, J. and Ghemawat, S. 2010. MapReduce: A flexible processing tool. *Commun. ACM* 53(1), 72–77. DOI: 10.1145/1629175.1629198.
- EMC Education Services. 2015. *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*. Indianapolis: John Wiley & Sons.
- European Commission. 2012. *A Study on Authentication and Authorisation Platforms for Scientific Resources in Europe: FINAL REPORT A Study Prepared for the European Commission DG Communications Networks, Content & Technology*. Advancing Technologies and Federating Communities (2011/0056).
- European Grid Initiative. 2016. Federated Cloud: IT Infrastructure, Your Way. Accessed June 01, 2016. <http://www.egi.eu/solutions/fed-cloud/>
- Fischer, G. 2012. Context-aware systems: “The ‘right’ information, at the ‘right’ time, in the ‘right’ place, in the ‘right’ way, to the ‘right’ person.” *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 287–294.
- Freiknecht, J. 2014. *Big Data in der Praxis. Lösungen mit Hadoop, HBase und Hive; Daten Speichern, Aufbereiten, Visualisieren*. Munich: Hanser.
- Gartner, Inc. 2015. Gartner - IT-Glossary Big Data. <http://www.gartner.com/it-glossary/big-data/>
- Ghemawat, S., Gobiuff, H., and Leung, S. 2003. The Google file system. *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. The Sagamore, Bolton Landing, Lake George. New York. USA. Association for Computing Machinery. *Operating Systems Review*. v. 37. no. 5 Dec. 2003.
- Google, Inc. 2016. Products & Services: Run Your Application Using the Same Technology and Tools Used at Google. Accessed May 07, 2016. <http://cloud.google.com/products>
- Holoman, J. and Gwen, S. 2014. Apache Kafka for Beginners. Accessed May 11, 2016. <http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners>
- Hortonworks Inc. 2016. Apache Falcon. Accessed May 13, 2016. <http://hortonworks.com/apache/falcon/>

- Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. 2015. *Learning Spark: Lightning-Fast Big Data Analytics* (1st ed.). O'Reilly Media, Inc.
- Kaufmann, M. 2016. *Towards a Reference Model for Big Data Management*. Research Report, Faculty of Mathematics and Computer Science, University of Hagen, retrieved July 15, 2016 from: https://ub-deposit.fernuni-hagen.de/receive/mir_mods_00000583.
- Kuhlen, R. 2004. *Informationsethik: Umgang mit Wissen und Information in elektronischen Räumen*. UTB 2454: Medien- und Kommunikationswissenschaft. Konstanz: UVK-Verl.-Ges.
- Landset, S., Khoshgoftaar, T. M., Richter, A. N., and Hasanin, T. 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1): 5–7. <http://doi.org/10.1186/s40537-015-0032-1>
- Lehner, F. 2014. *Wissensmanagement: Grundlagen Methoden und technische Unterstützung*. 5th ed. München: Hanser.
- Liu, S., Smith, M. H., Tuck, S., Pan, J., Alkurajji, A., and Jayawickrama, U. 2015. Where can knowledge-based decision support systems go in contemporary business management—A new architecture for the future. *JOEBM* 3(5): 498–504. DOI: 10.7763/JOEBM.2015.V3.235.
- Muise, A. 2014. Introduction to Apache Falcon: Data Governance for Hadoop. Accessed May 12, 2016. <http://de.hortonworks.com/blog/introduction-apache-falcon-hadoop/>
- Nonaka, I. and Takeuchi, H. 1995. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, Oxford University, New York.
- Siemens. 2014. New services for online condition monitoring of industrial plants. Available online at [http://www.siemens.com/press/en/pressrelease/?press=/en/pressrelease/2014/industry/customer-services/i2014032505.htm&content\[\]=ICS&content\[\]=DF](http://www.siemens.com/press/en/pressrelease/?press=/en/pressrelease/2014/industry/customer-services/i2014032505.htm&content[]=ICS&content[]=DF). Checked on August 11, 2016.
- Singh, D. and Reddy, K. C. 2014. A survey on platforms for big data analytics. *Journal of Big Data*, 1: 8. <http://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0008-6>
- Tableau Software. 2016. Tableau Desktop. Accessed May 05, 2016. <http://www.tableau.com/products/desktop>
- Thomas, J. J. and Cook, K. A. 2006. A visual analytics agenda. *IEEE Computer Graphics and Applications* 26(1): 10–13. DOI: 10.1109/MCG.2006.5.
- Upadhyay, S. and Grant, R. 2013. 5 Data Scientists Who Became CEOs—and are Leading Thriving Companies. Accessed June 09, 2016. <http://venturebeat.com/2013/12/03/5-data-scientists-who-became-ceos-and-are-leading-thriving-companies/>
- Veit, D., Clemons, E., Benlian, A., Buxmann, P., Hess, T., Kundisch, D., Leimeister, J. M., Loos, P., and Spann, M. 2014. Business models—An information systems research agenda. In: *Business & Information Systems Engineering*. Available at SSRN: <https://ssrn.com/abstract=2471635>
- Zhongzhi, S. 1987. Knowledge-based decision support system. *Journal of Computer Science and Technology* 2(1): 22–29. DOI: 10.1007/BF02943314.

10

Big Data Management Tools for Hadoop

V. P. Lijo, Lydia J. Gnanasigamani, Hari Seetha, and B. K. Tripathy

CONTENTS

10.1 Introduction.....	199
10.2 Hadoop Ecosystem.....	200
10.2.1 Apache Sqoop: “SQL to Hadoop and Hadoop to SQL”.....	200
10.3 Import Tool.....	200
10.4 Export Tool.....	201
10.4.1 Apache Flume.....	201
10.4.2 Apache Pig.....	202
10.4.3 Apache Hive.....	203
10.4.4 Apache HBase.....	204
10.5 Related Tools.....	204
10.5.1 Apache Oozie.....	204
10.5.2 Apache Zookeeper.....	205
10.5.3 Apache Ranger.....	207
10.5.4 Apache Tez.....	208
10.5.5 Apache Spark.....	208
10.5.6 Apache Mahout.....	209
10.5.7 Hue.....	209
10.6 Cassandra.....	209
10.6.1 Architecture.....	209
10.6.2 Data Model.....	210
10.6.3 Cassandra Query Language.....	210
10.6.4 Shell Commands.....	211
10.6.5 CQL Data Definition Commands.....	211
10.6.6 CRUD Operations.....	212
References.....	213

10.1 Introduction

Big Data are a large amount of data that are being generated from social networks, mobile devices, sensors, etc. It was a great challenge to store and process Big Data before the advent of Hadoop. Understanding Big Data drives the growth of any company (White, 2015). This is now possible with Hadoop. Hadoop is a set of libraries that provide methods to access and analyze these data in a fast and efficient manner. There are many flavors of Hadoop available to us. Apache Hadoop is open source, and a lot of projects have been developed surrounding Apache Hadoop (Lam, 2010). Cludera Hadoop is also available as

open source. It uses spark for in-memory processing. Other versions of Hadoop are available from IBM, Hortonworks, MapR, and TerraData. Hadoop is not a single component but a collection of many components that provide specialized services to users. These components collectively are called Hadoop ecosystem. The tools for data ingestion—Sqoop and Flume, service programming—Thrift and Zookeeper, task scheduling—Oozie, and machine learning—Mahout enrich the functionalities of Hadoop.

In this chapter, we discuss about Hadoop components, their architecture, and some of the major tools that make the Hadoop ecosystem. The important technology/technical terms used in this chapter are defined in the “Key Terminology & Definitions” section.

10.2 Hadoop Ecosystem

Hadoop ecosystem includes a lot of tools that address specific needs of users (Thinkbiganalyticscom, 2016). All these tools operate on top of Hadoop and leverage Hadoop’s parallelism to their own advantage. Below, we discuss a set of tools that make Hadoop even more powerful.

10.2.1 Apache Sqoop: “SQL to Hadoop and Hadoop to SQL”

Day-to-day transactions in major supermarkets and banks are one of the major sources of Big Data. These transaction data are traditionally stored in RDBMS like MySQL, Oracle, PostgreSQL, etc. When people started analyzing Big Data with Hadoop, a tool was required to efficiently import data residing on RDBMS into Hadoop. This is where Sqoop lies in the Hadoop ecosystem, facilitating the bulk transfer of data from structured data stores like RDBMS to Hadoop and vice versa (Apache Software Foundation, 2016c).

Sqoop contains two subtools (Intel, 2013): the import tool and the export tool (Figures 10.1 and 10.2).

10.3 Import Tool

The import tool is used to import individual tables from RDBMS into Hadoop. A table in RDBMS contains many rows, each of which, in Hadoop, is considered as a record. The records are stored as text files that have text data (White, 2015). The records can also be

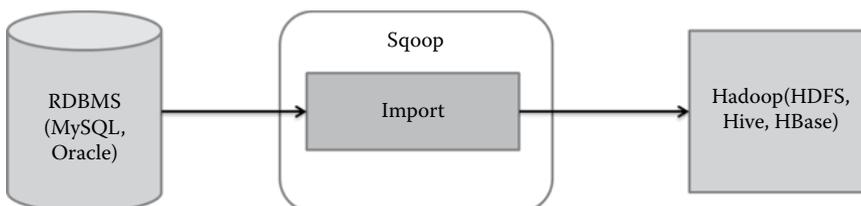


FIGURE 10.1
Sqoop import.

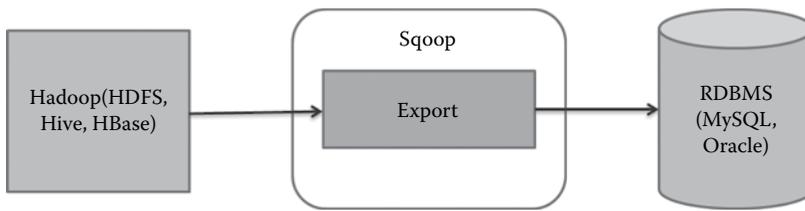


FIGURE 10.2
Sqoop export.

stored as binary data in sequence files. It is not always necessary to import the entire table into HDFS. If we require only part of the table, then the same can be imported by specifying the condition based on which the tools should select the required records.

10.4 Export Tool

The export tool does the reverse job of import tool. It is used to export the records from HDFS into some RDBMS. The records or the files in HDFS are transformed into rows in a table in RDBMS. The delimiter for every column in the table can be set as user-specified value. Before the records from HDFS are exported to an RDBMS, the table should be already created and present prior to the transfer.

10.4.1 Apache Flume

Apache flume is a tool that is used to collect and aggregate streaming data from various different sources and transport them to a centralized data store. Much of the data that need to be analyzed are produced by many different sources like social networking sites, application servers, and cloud servers. Stream data generally consist of two types of data: event data and log data. Event data are huge volumes of data that are produced by social networking sites like twitter, Facebook, etc. and e-commerce sites like Flipkart, Amazon, etc. These event data are used to analyze and find the current trends and customer behavior. Log data are a list of actions that happen in a web server. For example, every request made to the server can be logged. Every failed response can be logged, and all the exceptions thrown by the server can also be written in the log file. These data are very huge in volume, but on analyzing such data, we can get information on performance, the kind of requests the web server gets the most, and hardware and software failures if any.

Figure 10.3 shows the working of Flume. Both the log and event data are very huge in volume and need to be moved to Hadoop for analysis. Flume is used to move these data into HDFS or HBase at higher speeds. Flume supports a large set of different sources and destinations or sinks. It can transfer the data immediately into Hadoop. Flume can act as a mediator between data generators and sinks when the rate of data production is much higher than the speed at which it can be written to the destinations. More importantly, it is scalable, fault tolerant, and customizable.

The Flume architecture is shown in Figure 10.4. Each agent is a daemon JVM (Java Virtual Machine) process. Agent nodes are generally installed on the data generator side, that is, the machines that generate the log and event data. The collector node, which is also

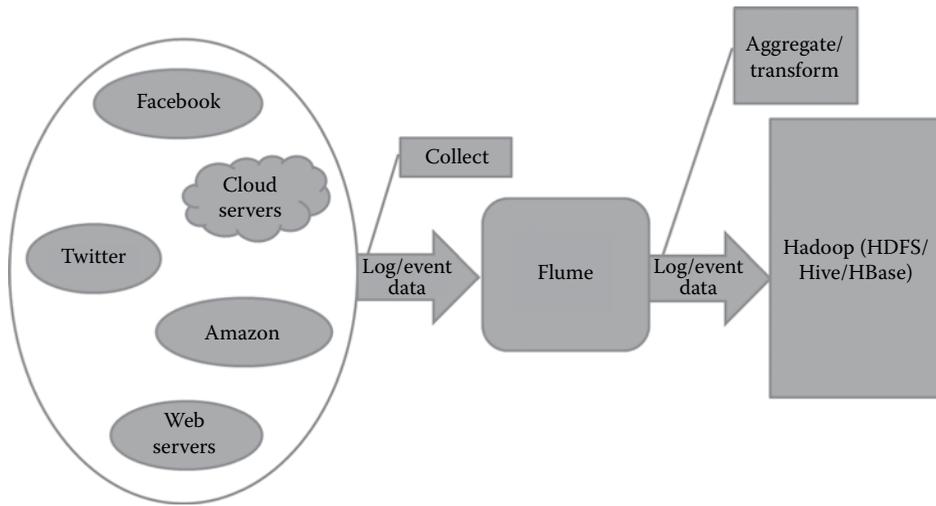


FIGURE 10.3
Working of flume.

an agent node, collects or aggregates data from the agent nodes and sends it to the sink nodes (storage nodes).

Both Sqoop and Flume are ETL (Extract, Transform, Load) (Intel, 2013) tools.

10.4.2 Apache Pig

Apache Pig is a platform that is used for analyzing Big Data sets using a high-level language called Pig Latin. Pig Latin is an abstraction over MapReduce, and the script written

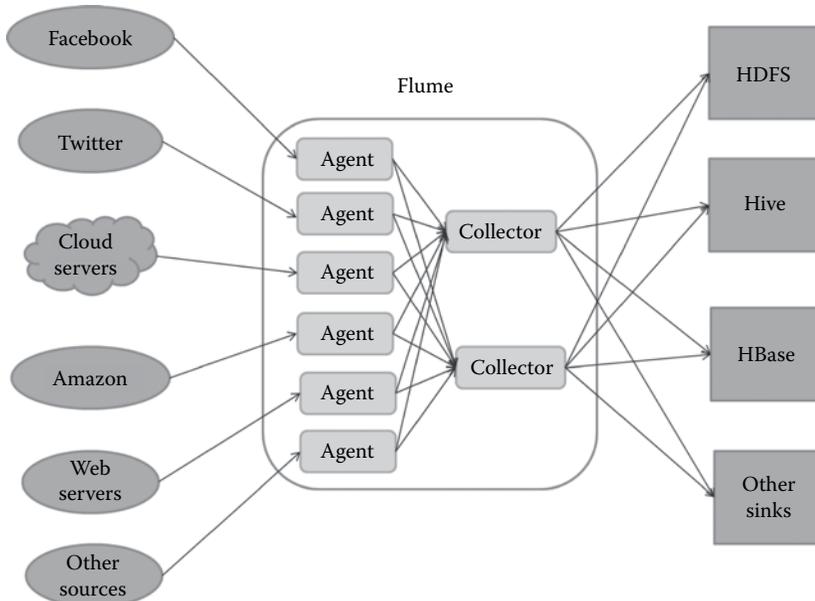


FIGURE 10.4
Flume architecture.

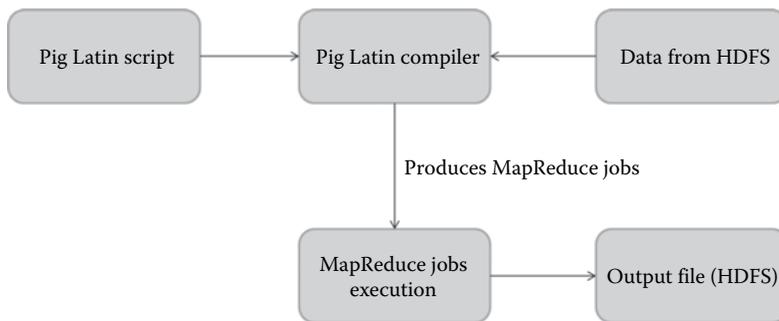


FIGURE 10.5
Pig components.

in Pig Latin gets converted into a series of MapReduce jobs by the Pig Engine (Olston et al., 2008) (Stonebraker et al., 2010). About only 16 lines of Pig Latin code is required to do what we do with 200 lines of Java code (Apache Software Foundation, 2016e). Pig Latin is a procedural language that has syntax similar to that of SQL. It is very easy to achieve parallel processing using Pig. Complex task that consists of many transformations is modeled as multiple data flow sequences separately, which are easier to modify and maintain. Pig Latin allows creation of User-Defined Functions (UDFs) for special functionalities. Apache pig is generally used today for performing ad hoc processing, time-sensitive data, quick prototyping, and data processing on search platforms. The components of Pig are shown in [Figure 10.5](#).

10.4.3 Apache Hive

Apache Hive is a data warehouse tool that is used to process structured data in Hadoop. It allows users to write SQL-like queries to process the structured data stored in HDFS (Apache Software Foundation, 2016f). The language provided by Hive is called HiveQL or HQL. Hive was developed by Facebook; it is now under Apache Software Foundation as an open source (Thusoo et al., 2010). It is fast, extensible, and reliable. It is designed for Online Analytical Processing (OLAP). Hive provides a web-based user interface, a command-line tool, and a JDBC driver to enable users to connect to it.

Hadoop provides easy access to data via SQL and enables reporting, analysis, and ETL functions on data that are stored on a distributed data store. It can access files stored on both HDFS and HBase. Hive queries can be executed using MapReduce, Apache Spark, or Apache Tez. Hive has not been designed for Online Transaction Processing (OLTP), but rather as a software for performing data warehousing tasks.

The Hive components are shown in [Figure 10.6](#). The Meta store service of Hive stores the metadata of the Hive tables, partitions, columns, their data types, and their mapping in HDFS in an RDBMS. The access to this information is provided through the Meta store API. The default database for metastore is Apache Derby. Optionally, other RDBMS can also be used as Meta store. Some of the supported databases for metastore are MySQL, Postgres, Oracle, and MS SQL Server.

The Execution Engine processes the HiveQL queries using one of the following: Apache Tez, Apache Spark, or MapReduce. Hive provides support to UDFs to manipulate strings, dates, and other data-mining manipulations.

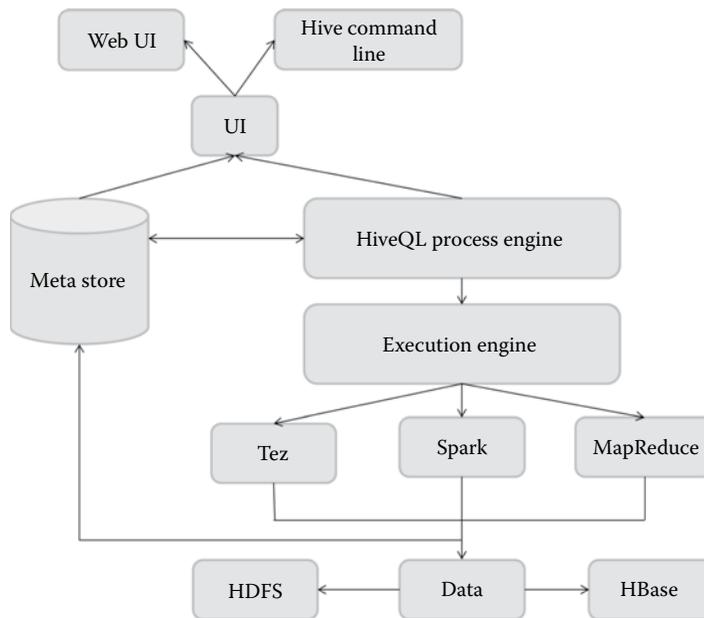


FIGURE 10.6
Hive components.

10.4.4 Apache HBase

Apache HBase provides random real-time access to tables that contain billions of rows. HBase was modeled after Google Big Table (Apache Software Foundation, 2016d). It is designed to provide random read/write access to large structured data. HBase is a distributed nonrelational database that works on top of Hadoop and HDFS. HBase is a column-oriented database just like many other nonrelational databases are (Allene and Righini, 2016). The tables are sorted by rows, each row is a group of column families, each column family is a group of columns, and each column is a group of key–value pairs. It is good for both semistructured and structured data. It is linearly scalable and is used for wide tables. It provides automatic sharing of tables and automatic failover support between region servers.

The HBase architecture is shown in [Figure 10.7](#). The Master server assigns regions to region servers with the help of Zookeeper. It handles load balancing, does schema changes, and is also responsible for creation of tables and column families. Regions are tables that are split and spread across the region servers. The region server handles the read/write requests for the regions under it and also decides upon the region size based on some user set threshold.

10.5 Related Tools

10.5.1 Apache Oozie

Apache Oozie is a workflow scheduling system that is used to manage different Hadoop jobs. Using Oozie, various programs can be pipelined in the order which we desire to

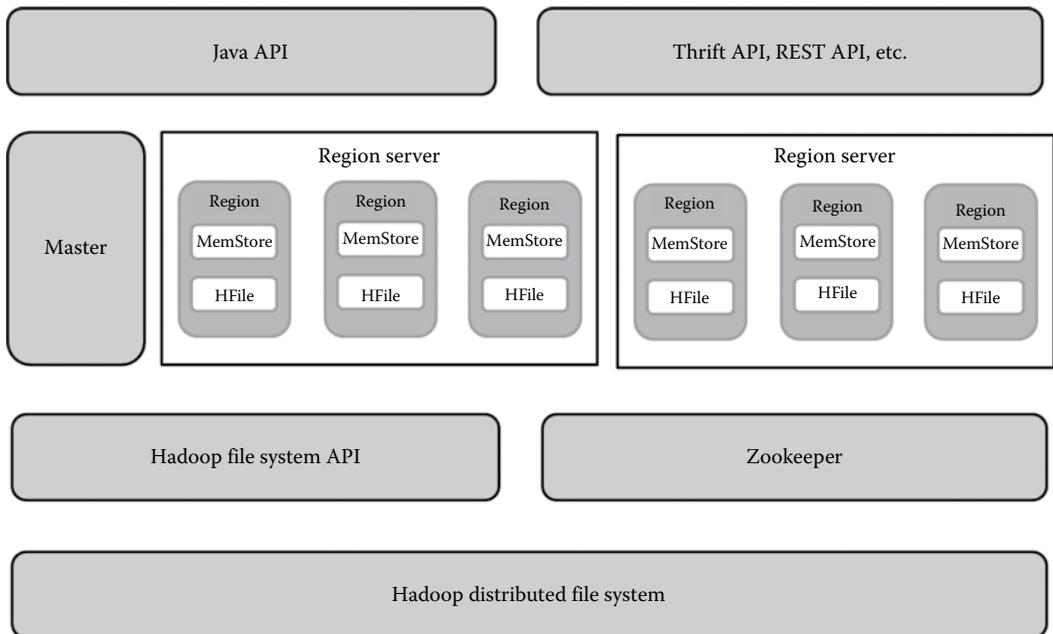


FIGURE 10.7
HBase architecture.

work on top of Hadoop. It supports jobs like MapReduce, Hive, Pig, Sqoop, etc. It can also include java and shell scripts in its workflow (Apache Software Foundation, 2016g).

Workflow in Oozie can be defined as collection of control nodes and action nodes in a Directed Acyclic Graph (DAG) (Singh, 2015). Control nodes are the start and end nodes, and the decision nodes decide the path the workflow should take. Action nodes are the different types of jobs Oozie supports. There are three types of workflows in Oozie:

- Oozie workflow jobs: Represented as DAGs to specify the actions to be executed.
- Oozie coordinator jobs: Repeated Oozie workflow jobs that are triggered by time and data availability.
- Oozie bundle: Package of multiple coordinator and workflow jobs.

Hue editor is the popular editor for creating Oozie workflows. It provides a GUI with drag and drop facility. Oozie Eclipse Plugin (OEP) can also be used for creating and editing workflows graphically.

Figure 10.8 shows the architecture of Oozie. HCatalog is a centralized metadata management and sharing service for Apache Hadoop. It allows for a unified view of all data in Hadoop clusters and allows diverse tools, including Pig and Hive, to process any data elements without needing to know physically where in the cluster the data are stored.

10.5.2 Apache Zookeeper

Apache Zookeeper is a distributed coordination service to manage a large number of servers as present in a Hadoop cluster (Hortonworkscom, 2015). Zookeeper also provides services like configuration management, naming service, synchronization, cluster

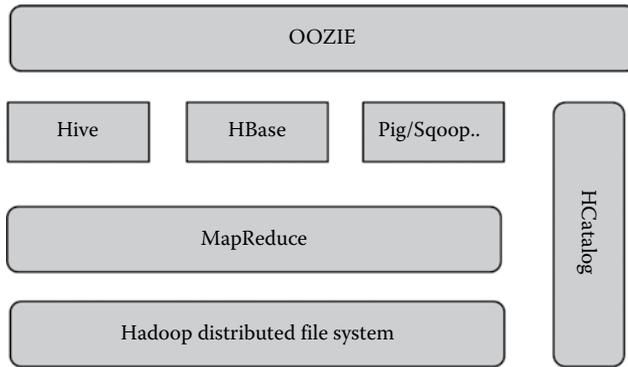


FIGURE 10.8
Oozie architecture.

management, data registry, and notification. If a programmer is to take care of all these in a code that he/she writes for distributed processing, it is possible that there might be a lot of bugs. Even if there are not initially, the changes in code would inevitably introduce bugs which might lead to RACE conditions. But Zookeeper lets the programmer concentrate on developing the core logic and takes care of everything else needed for distributed processing. Zookeeper was initially developed at Yahoo, but now it is an open-source project under Apache. It is used by Yahoo, eBay, Solr, and Rackspace, among other companies. Zookeeper is used to find the available nodes and also to find server failures and initiate failovers.

From Figure 10.9, Ensemble is a group of Zookeeper servers. Leader node is the one that is responsible for automatic failovers in case of node failures (Hunt et al., 2010). A leader is elected at the start of service. Follower is the node that follows leader’s instruction. Each client is connected to only one server, and all the read requests from that client are serviced by that server. When a server receives a write request, it is sent to the “Leader.” The

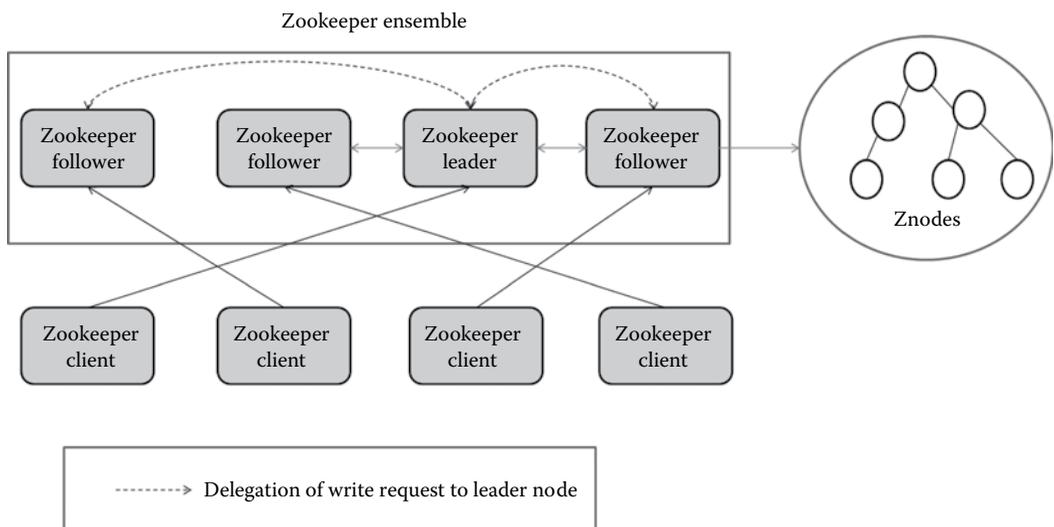


FIGURE 10.9
Zookeeper architecture.

leader then delegates the task to available nodes. So, the write requests are always serviced only by the Leader, and hence, they take more time than read requests.

Zookeeper has a file system-like data model, referred to as “znode.” Just like a file system has directories, znodes are also directories, and they can have data associated with them. The znode can be referenced through absolute path separated with a slash. Each server in the ensemble stores the Znode hierarchy in the memory, which makes the response quick and scalable as well. There are three types of znodes: the Persistence znode, which is alive even after the client is disconnected; the Ephemeral znode, which is active only as long as the client is connected and is deleted automatically once the client disconnects; and the Sequential znode, which has a 10-digit number attached to it at the end. The Sequential znode can either be persistence znode or ephemeral node.

10.5.3 Apache Ranger

Ranger is a framework that enables, monitors, and manages comprehensive security across the Hadoop platform (Apache Incubator™, 2016). It provides a centralized security administration using REST APIs and UI. Ranger provides support for different types of authorization control like role-based control and attributed-based control. It provides standardized security across all Hadoop components. Using the administration console of Ranger, we can easily create and manage policies for accessing a resource like file, folder, or database access for a particular user or group of users. We can also enable audit tracking for having more control over the environment. Ranger supports authorization for Hive, HBase, Solr, and many more Hadoop components.

Ranger architecture is shown in Figure 10.10. The Ranger portal is the interface for security administration. The admin can create and update policies using the portal. These policies are then stored in the policy data store. The policy data store can be HDFS itself or any other RDBMS (Hortonworkscom, 2015). Ranger plugins are nothing but lightweight java components which are embedded within the Hadoop components on which security policies need to be enforced (White, 2015). The plugins pull policies from Ranger Policy Server.

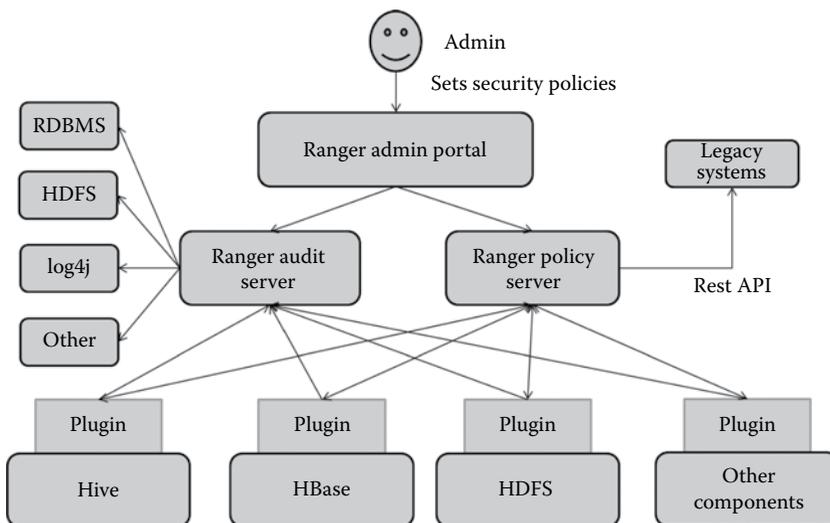


FIGURE 10.10
Ranger architecture.

Every request sent to component is intercepted by the plugin and checked against the policy. Plugins also send audit data; data that is collected from the user request to Ranger Audit Server in a separate thread. These data are then stored in HDFS, RDBMS, or log4j.

10.5.4 Apache Tez

Apache Tez is a framework used for building batch and interactive data processing applications on Apache YARN in Hadoop (Apache Software Foundation, 2015a). Tez provides expressive dataflow definitions. These dataflow are modeled as DAGs for processing data. It has simplified deployment procedure. Tez allows Hive and Pig to run complex DAG of tasks. The data that were earlier processed using multiple MapReduce jobs are done by a single Tez job now. This improves the performance many times over MapReduce. It also allows dynamic data flow decisions. Tez also provides high horizontal scalability.

10.5.5 Apache Spark

Apache Spark is a cluster computing framework. It was developed at the University of California, Berkley and then donated to Apache Software Foundation. It provides an interface for programming entire clusters with data parallelism and fault tolerance (Apache Software Foundation, 2016b). Spark provides application programming interface based on a data structure called the resilient-distributed data set (RDD) (Zaharia et al., 2012). RDD is a read-only multiset of data items distributed over a cluster of machines, maintained in a fault-tolerant way. One of the important features of Spark is its in-memory cluster computing (Zaharia et al., 2010) which increases the processing speed of an application. Spark can work on different workloads such as batch processing, iterative algorithms, interactive queries, and streaming. Spark provides built-in support for many languages such as Java, Python, and Scala. Spark also supports Machine Learning and Graph algorithms.

Figure 10.11 shows the Spark components. Spark Core is the execution engine that all other functionalities of Spark platform are built upon (Apache Software Foundation, 2016b). It provides in-memory computing and referencing external storage systems. Spark SQL introduces a new abstract data structure, the SchemaRDD. It is used to provide support for structured and semistructured data. Spark Streaming enables analytics for stream data. It reads mini batches of data and performs RDD transformations on them. MLlib is the distributed machine learning framework and is about nine times faster than Apache

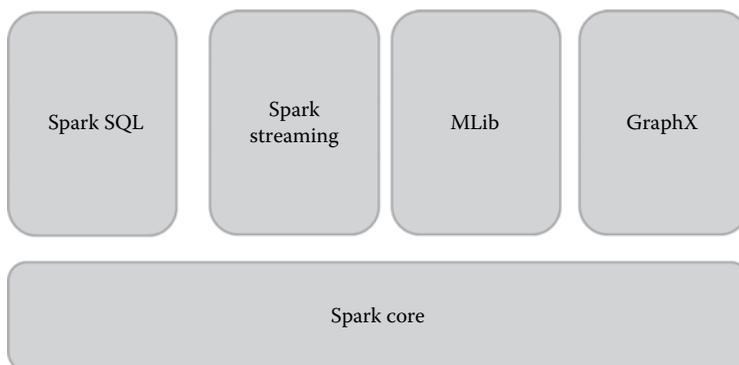


FIGURE 10.11
Spark components.

Mahout with MapReduce. GraphX is a distributed graph processing framework that provides an API for expressing graph computation.

10.5.6 Apache Mahout

Apache Mahout provides an extensible and scalable environment for building scalable machine learning algorithms (Apache Software Foundation, 2016a). It also has a lot of built-in algorithms for Spark and Scala. The popular machine learning techniques implemented on Mahout are Classification and Clustering. Mahout analyzes large sets of data quickly, and it works on top of Hadoop, so it is scalable. Many companies such as Yahoo, Twitter, Facebook, and LinkedIn use Mahout for pattern mining, modeling, as recommender engine, and other analysis purpose.

10.5.7 Hue

Hue is a web interface that supports Hadoop and its ecosystem. Hue aggregates most of the common Hadoop components in a common interface. The users can just use Hadoop without using a command-line utility or worrying about the internal complexity (Hue Team, 2014). It serves as file browser for HDFS; as a job browser for MapReduce; as a browser for HBase and zookeeper; as a query editor for Hive, Impala, and pig; as editor for Oozie and Sqoop; and also as a dashboard for Oozie and Solr. It also provides Spark editor.

10.6 Cassandra

Cassandra is a distributed database from Apache. It is scalable and can manage huge amounts of structured data (Apache Software Foundation, 2015b). It provides high availability with no single point of failure. Cassandra has revolutionized the way enterprises deal with large volumes of data in an efficient and 100% fault-tolerant way. It was originally developed by Facebook and then released as open source (Mvdironacom, 2016). It is a java-based NOSQL database able to handle critical deployments across servers.

NoSQL databases provide mechanisms to store and retrieve other than the tabular relations in RDBMS. These databases are schema less, handle huge amounts of data, support easy replication, and also provide horizontal scaling.

Cassandra is a column-oriented database. Its data model is based on Google's Bigtable. It was created at Facebook. It can support structured, semistructured, and unstructured data. It can accommodate dynamic changes to the data structures as and when needed. It provides transaction support by supporting the ACID properties. It can write very fast (Rabl et al., 2012) and can store hundreds of terabyte of data.

10.6.1 Architecture

Cassandra has peer-to-peer distributed system across its nodes, and data are distributed among all the nodes in a cluster. All the nodes in a cluster play the same role. Each node is independent of and at the same time interconnected to other nodes. Each node in a cluster can accept read and write requests, regardless of where the data are actually located in the cluster. When a node goes down, read/write requests can be served from other nodes in the network.

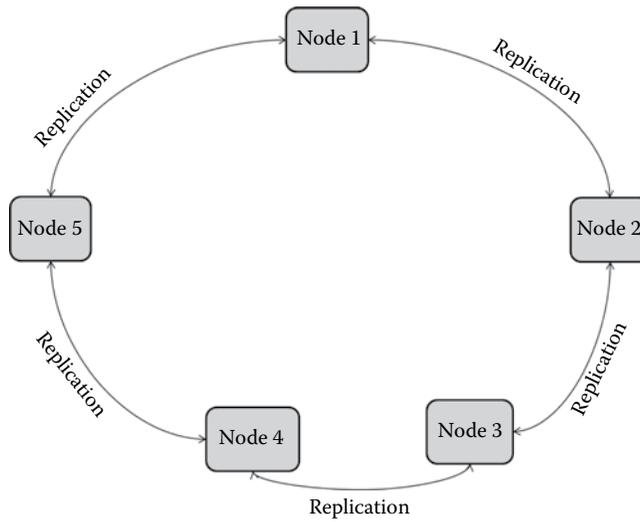


FIGURE 10.12
Cassandra data model.

In Cassandra, data are replicated in one or more nodes. If it detects old value being returned by some of the nodes, it returns the most recent value to the client and updates all the stale values by doing a read–repair.

10.6.2 Data Model

Cassandra database is distributed across several machines that operate together as a cluster. For handling failures, every node contains a replica, and in case of failures, the replica takes charge. Cassandra arranges the nodes in a ring format and assigns data to them, as shown in [Figure 10.12](#).

10.6.3 Cassandra Query Language

Cassandra by default provides a prompt Cassandra query language shell (cqlsh) for users to communicate with it. Using this cqlsh shell, we can write and execute Cassandra Query Language (CQL). Using cqlsh, we can define schema, insert data, and execute a query.

To start cqlsh, type cqlsh in the terminal, and we will get the cqlsh prompt as below:

```

[cqlsh 5.0.1 | Cassandra 2.1.3 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh>
  
```

10.6.4 Shell Commands

- HELP**—Displays help topics for all cqlsh commands.
- CAPTURE**—Captures the output of a command and adds it to a file.
- CONSISTENCY**—Shows the current consistency level, or sets a new consistency level.
- COPY**—Copies data to and from Cassandra.
- DESCRIBE**—Describes the current cluster of Cassandra and its objects.
- EXPAND**—Expands the output of a query vertically.
- EXIT**—Using this command, you can terminate cqlsh.
- PAGING**—Enables or disables query paging.
- SHOW**—Displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.
- SOURCE**—Executes a file that contains CQL statements.
- TRACING**—Enables or disables request tracing.

10.6.5 CQL Data Definition Commands

- CREATE KEYSPACE**—Creates a KeySpace in Cassandra.

A keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node.

Syntax: CREATE KEYSPACE <identifier> WITH <properties>
 CREATE KEYSPACE "KeySpace Name" WITH replication = {'class': 'Strategy name', 'replication_factor' : 'No.Of replicas'};

The replication option is to specify the *Replica Placement strategy* and the number of replicas wanted. The following table lists all the replica placement strategies.

Strategy Name	Description
Simple strategy	Specifies a simple replication factor for the cluster.
Network topology strategy	Using this option, you can set the replication factor for each datacenter independently.
Old network topology strategy	This is a legacy replication strategy.

```
[cqlsh 5.0.1 | Cassandra 2.1.3 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh> create keyspace example with replication ={'class':'SimpleStrategy','replication_factor':3};
```

- ALTER KEYSPACE**—Changes the properties of a KeySpace.

ALTER KEYSPACE can be used to alter properties such as the number of replicas and the durable_writes of a KeySpace.

Syntax:

```
ALTER KEYSPACE "KeySpace Name" WITH replication = {'class': 'Strategy
name', 'replication_factor' : 'No.Of replicas'};
```

DROP KEYSPACE—Removes a KeySpace.

You can drop a KeySpace using the command DROP KEYSPACE.

Syntax: DROP KEYSPACE "KeySpace name"

10.6.6 CURD Operations

CREATE—Adds columns for a row in a table.

You can insert data into the columns of a row in a table using the command INSERT. Given below is the syntax for creating data in a table.

```
INSERT INTO <tablename> (<column1 name>, <column2 name>...) VALUES
<value1>, <value2>...) USING <option>
```

UPDATE—Updates a column of a row.

UPDATE is the command used to update data in a table. The following keywords are used while updating data in a table:

Where—This clause is used to select the row to be updated.

Set—Set the value using this keyword.

Must—Includes all the columns composing the primary key.

While updating rows, if a given row is unavailable, then UPDATE creates a fresh row. Given below is the syntax of UPDATE command:

```
UPDATE <tablename>
SET <column name> = <new value>
<column name> = <value>...
WHERE <condition>
```

READ—Executes multiple DML statements at once.

SELECT clause is used to read data from a table in Cassandra. Using this clause, you can read a whole table, a single column, or a particular cell. Given below is the syntax of SELECT clause.

```
SELECT FROM <tablename>
```

Using WHERE clause, you can put a constraint on the required columns. Its syntax is as follows:

```
SELECT FROM <table name> WHERE <condition>;
```

DELETE—Deletes data from a table.

You can delete data from a table using the command DELETE. Its syntax is as follows:

```
DELETE FROM <identifier> WHERE <condition>;
```

References

- Allene, B. and Righini, M. 2016. Better Performance for Big Data. Retrieved 5 June, 2016, from <http://www.intel.in/content/dam/www/public/us/en/documents/white-papers/big-data-financial-services-better-performance-big-data-whitepaper.pdf>
- Apache Incubator™. 2016. Apache Ranger. Retrieved 10 June, 2016, from <http://ranger.apache.org/>
- Apache Software Foundation. 2015a. TEZ. Retrieved 1 June, 2016, from <http://tez.apache.org>
- Apache Software Foundation. 2015b. Cassandra. Retrieved 1 June, 2016, from <http://cassandra.apache.org>
- Apache Software Foundation. 2016a. Mahout. Retrieved 2 June, 2016, from <http://mahout.apache.org>
- Apache Software Foundation. 2016b. Apache Spark-Lightning-Fast Cluster Computing. Retrieved 1 June, 2016, from <http://spark.apache.org>
- Apache Software Foundation. 2016c. Sqoop. Retrieved 5 June, 2016, from <http://sqoop.apache.org>
- Apache Software Foundation. 2016d. Welcome to Apache HBase™. Retrieved 2 June, 2016, from <http://hbase.apache.org>
- Apache Software Foundation. 2016e. Apache Pig. Retrieved 10 June, 2016, from <http://pig.apache.org>
- Apache Software Foundation. 2016f. Apache Hive TM. Retrieved 10 June, 2016, from <http://hive.apache.org>
- Apache Software Foundation. 2016g. Apache Oozie Workflow Scheduler for Hadoop. Retrieved 5 June, 2016, from <http://oozie.apache.org>
- Hortonworkscom. 2015. A Holistic Approach to a Secure Data Lake. Retrieved 1 June, 2016, from http://hortonworks.com/wp-content/uploads/2015/07/Security_White_Paper.pdf
- Hue Team. 2014. Hue—Hadoop User Experience—The Apache Hadoop UI. Retrieved 2 June, 2016, from <http://gethue.com>
- Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. 2010. ZooKeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference* (Vol. 8, p. 9).
- Intel. 2013. <http://hadoopintel.com>. Retrieved 3 May, 2016, from <http://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf>
- Lam, C. 2010. Hadoop in action. Manning Publications Co.
- Mvdironacom. 2016. Mvdironacom. Retrieved 2 June, 2016, from <http://perspectives.mvdironacom/2008/07/facebook-releases-cassandra-as-open-source/>
- Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. 2008. Piglatin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1099–1110). ACM.
- Rabl, T., Gómez-Villamor, S., Sadoghi, M., Muntés-Mulero, V., Jacobsen, H. A., and Mankovskii, S. 2012. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12), 1724–1735.
- Singh, J. J. 2015. *Apache Oozie Essentials*, Packt Publishing Ltd., Birmingham, UK.
- Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., and Rasin, A. 2010. MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, 53(1), 64–71.
- Thinkbiganalyticscom. 2016. *Thinkbiganalyticscom*. Retrieved 4 June, 2016, from 1.http://thinkbiganalytics.com/leading_big_data_technologies/hadoop/
- Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., and Murthy, R. 2010. Hive-a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)* (pp. 996–1005). IEEE.
- White, T. 2015. *Hadoop: The Definitive Guide*, 4th edition. O'Reilly Media, California, USA.
- Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. 2010. Spark: Cluster computing with working sets. *HotCloud*, 10, 10-10.
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... and Stoica, I. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (pp. 2–2). USENIX Association, San Jose, CA.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

11

Realization of Optimized Clustering Algorithm on RHadoop

Poonam Ghuli, Raksha Shenoy, Shankararama Sharma R, and Rajashree Shettar

CONTENTS

11.1	Introduction to Clustering Algorithms	216
11.1.1	Technology Background on Clustering Process	216
11.1.2	Hard Partitional Clustering Algorithms	217
11.1.3	Hierarchical Clustering Algorithms	217
11.1.4	Fuzzy Clustering Algorithms	217
11.2	Distributed Programming Paradigm	218
11.2.1	MapReduce Programming Model.....	218
11.2.2	Hadoop Architecture	219
11.2.3	RHadoop Platform.....	221
11.3	Improvisations of K-Means Clustering Algorithm.....	221
11.3.1	K-Means++ Algorithm.....	222
11.3.2	Overview of Parallel K-Means++ Algorithm	222
11.3.3	Research Challenges.....	223
11.3.4	Contributions toward Proposed Work	224
11.4	Proposed Methodology.....	225
11.4.1	Initial Centroid Selection for Serial K-Means++	226
11.4.2	Initial Centroid Selection for Parallel K-Means++	226
11.4.3	Design of Iterative Clustering Module on RHadoop.....	226
11.4.4	Cluster Assignment Module on RHadoop.....	228
11.5	Results and Discussions	228
11.5.1	Dataset Used.....	230
11.5.2	Experimental Setup	230
11.5.3	Performance Evaluation.....	230
11.5.3.1	Size-Up Analysis for Distributed Serial K-Means++ and Parallel K-Means++ Algorithm	231
11.5.3.2	Total Time Taken to Converge by the Algorithms and Number of Iterations Required for Convergence.....	231
11.5.3.3	Evaluations of Clusters and Determining Optimal Number of Clusters.....	233
11.5.3.4	Visualization Results	234
	References.....	235

11.1 Introduction to Clustering Algorithms

The tremendously increasing data volumes require advanced techniques and tools to quickly perform analysis on large-scale datasets. Clustering (Jiang et al. 2004; Hruschka et al. 2009) is the most popular and commonly used technique for processing huge amount of data. On top of it, K-means is a well-accepted cluster analysis algorithm and is considered as a major research topic due to its simplicity and ability to parallelize naturally. This chapter discusses the clustering process and the various types of clustering techniques such as partitional, hierarchical, and fuzzy clustering algorithms.

11.1.1 Technology Background on Clustering Process

Clustering partitions data into groups that will disclose some interesting patterns that may be meaningful and helpful to make important decisions. Clustering has its applications in various fields such as knowledge discovery, pattern recognition, document categorization, web mining, etc. Clustering is an unsupervised classification method that works on unlabeled data. The clustering process as depicted in Figure 11.1 involves four steps as listed below:

- a. *Feature extraction*: Feature selection is choosing some useful features from a feature set. On the contrary, feature extraction uses some transformations such as PCA (principal component analysis), ICA (independent component analysis), etc. to obtain useful features from the original ones.
- b. *Algorithm design for clustering in a distributed paradigm*: This step corresponds to the selection of appropriate distance measure along with the clustering algorithm that can work on large-scale datasets and is easily parallelizable in a distributed environment.
- c. *Cluster validation*: This step provides preference to select proper indices to measure clustering quality. This will help in the assessment of final clusters obtained as a result of applying clustering algorithm on the dataset.
- d. *Interpretation of results*: Extracting knowledge from the provided input is the major goal of any clustering approach. Interpretation of the data partition is done by the experts in the relevant fields by using visualization tools such as scatter plot, ggplot, etc.

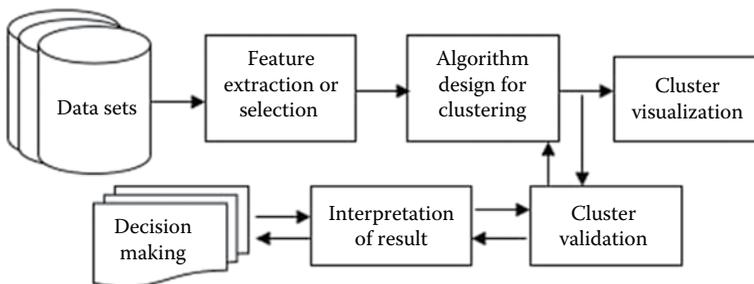


FIGURE 11.1

Clustering procedure. (From Xu, R. and D. Wunsch II, *IEEE Transactions on Neural Networks*, Vol. 16, 2005: 645–678.)

Clustering approaches are categorized into three major types (Jain et al. 1999; Xu et al. 2005): overlapping (nonexclusive), partitional, and hierarchical. In overlapping clustering, each data point may belong to more than one cluster. Furthermore, these overlapping partitions may generate soft or fuzzy data groups. In fuzzy clustering, each object may belong to one or more clusters with different degrees of membership. On the other hand, clustering allocates each data point to a single cluster during its operation and in its output. Mathematical representations of various categories of clustering are listed below in Equations 11.1 through 11.3.

Given a set of input data points, $X = \{x_1, x_2, \dots, x_j, \dots, x_N\}$, where $X_j = \{x_{j1}, x_{j2}, \dots, x_{jd}\}^T \in \mathcal{R}^d$, and each measure x_{ji} is a feature that represents an attribute, dimension, or variable.

11.1.2 Hard Partitional Clustering Algorithms

Hard partitional clustering of dataset X (given in Equation 11.1) is a collection $C = \{C_1, C_2, \dots, C_k\}$ of K disjoint data splits such that

$$C_i = \emptyset, \quad \cup_{i=1}^k C_i = X, \quad C_i \cap C_j = \emptyset, \quad i, j = 1, 2, \dots, k, \quad \text{and} \quad i \neq j \quad (11.1)$$

Here, $(k \leq N)$, where N is the total number of data points.

11.1.3 Hierarchical Clustering Algorithms

Hierarchical clustering constructs a tree-like nested structure $H = \{H_1, H_2, \dots, H_Q\}$, which divides the dataset X , as given below:

$$C_i \in H_m, \quad C_j \in H_l, \quad \text{and} \quad m > l \quad \text{imply} \quad C_i \in C_j \quad \text{or} \quad C_i \cap C_j = \emptyset \\ \text{for all } i; \quad j \neq i; \quad m, l = 1, 2, \dots, Q \quad (11.2)$$

11.1.4 Fuzzy Clustering Algorithms

Fuzzy clustering permits a data point to be part of all clusters with a degree of membership, $u_{ij} \in [0,1]$, which represents the membership coefficient of the j th object in the i th cluster and satisfies the two constraints as represented below:

$$\sum_{i=1}^c u_{ij} = 1, \quad \forall j \quad \text{and} \quad \sum_{j=1}^N u_{ij} < N, \quad \forall i \quad (11.3)$$

Clustering is a challenging task due to its unsupervised nature. This implies that the structural characteristics such as distribution of the data in terms of the number, volume, density, shape, etc. are unknown (Jain et al. 1999; Xu et al. 2005). These difficulties may be increased further as there will be need for handling data points represented by different types of attributes such as binary, discrete, continuous, and categorical; condition of dataset (whether the dataset is complete in all respect or values of some attributes are missing); and scale of measurement such as ordinal and nominal levels. Clustering (Kanungo et al. 2002; Pantel et al. 2003; Rokach and Maimon 2005) is a typical type of NP-hard grouping problem.

The major downside of K-means is its inability to select appropriate initial centroids to obtain optimal clusters. If no mechanism is used to choose initial centroids, the algorithm is unable

to obtain globally optimal solution. Thus, this chapter focuses on the study and test of the performance of scalable implementations of clustering algorithms such as serial K-means++ and parallel K-means++ using distributed framework such as RHadoop. The experiments conducted show how centroid initialization methods help to obtain optimal clusters.

11.2 Distributed Programming Paradigm

Currently, data are collected from social networks, sensors, digital images, sales transaction records, etc. The analysis of these data helps to reveal interesting patterns, market trends, co-relations, and useful business information that might provide valuable insights to users. To cope with this explosive increase in global data, there is a need of a scalable distributed (Guo et al. 2014) model for large-scale data-intensive computing. For using scalable distributed systems, one must ensure its reliability. To achieve reliability, the distributed system must be highly fault tolerant, highly available, consistent, scalable, and highly secure. Today, the memory of a single machine is incapable of storing these massive input data and processing them. To deal with this problem, computer experts have initiated parallelization of existing algorithms in a distributed environment. Moreover, the best alternative is to exploit the MapReduce (Dean and Ghemawat, 2004, 2008; Yoon et al. 2014) paradigm to solve special types of parallelizable or distributable problems. These problems involve processing and analyzing large datasets on a cluster of commodity components.

The major research challenge is to explore clustering algorithms on a distributed computing environment to process a huge dataset. The dataset can be large in three different ways. First, the dataset may consist of a large number of elements. Second, many features describe a particular element. Third, it may require determining several clusters. Clustering is an expensive task that involves numerous computations on large datasets for which distributed processing of data using different implementations of MapReduce (QiuHong et al. 2014) is an attractive solution. This section provides an overview on MapReduce programming model, Hadoop architecture, and RHadoop platform.

11.2.1 MapReduce Programming Model

Over the years, MapReduce has evolved into a simple, scalable, and fault-tolerant programming model that enables distributed processing of large data on a cluster of commodity nodes. MapReduce is a programming model whose main idea is to allow developers to focus on data processing algorithms and hide the data distribution details such as assigning, and scheduling of jobs, verifying job completion, managing data communication between the nodes, etc.

The MapReduce programming consists of the implementation of two functions: map and reduce. The mapper's job is to read the input line by line and split the data into small chunks of equal size. Usually, input data are stored in Hadoop Distributed File System (HDFS) in the form of a file. The input to map function is provided as a list of (key, value) pairs. Any machine learning problem that is expressed in summation form as shown in Equation 11.4 can be modeled as MapReduce tasks.

$$y = \sum (f(x)) \quad (11.4)$$

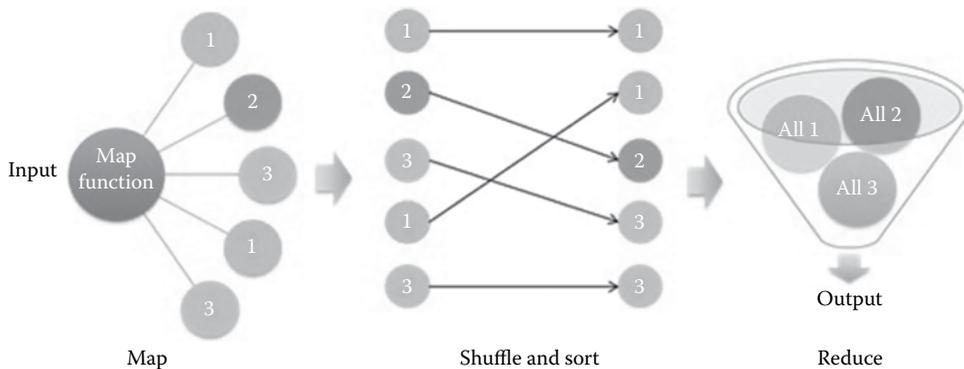


FIGURE 11.2
A typical MapReduce task flow.

Here, $f(x)$ is the map job and summation is the reduce job. Summation can be any task such as summing up, taking average, grouping, etc. A simple MapReduce task is illustrated in [Figure 11.2](#). In the MapReduce (Grolinger et al. 2014) programming paradigm, a job to be performed is specified in three stages (Karloff et al. 2010; Lee et al. 2012): map stage, sort and shuffle stage, and reduce stage. The map phase is responsible for processing input data and generates intermediate results in the form of key/value pairs. The intermediate key/value pairs are exchanged and merge-sorted on values based on key. Furthermore, all the values that are related to an individual key are given to the same reducer task.

In the reduce stage, the reducer is given as input all the values related to a single key K , and outputs key/value pairs by aggregating the results on the same key, K . All the map jobs are required to complete before any reduce job starts. Overall, the implementation of an algorithm in the MapReduce paradigm (Rao et al. 2012; Lee et al. 2012) can include several iterations of different map and reduce functions, executed one after another.

11.2.2 Hadoop Architecture

Apache Hadoop is a popular open-source Java implementation of MapReduce that comes with two packages. First, it consists of a data storage package called HDFS (Borthakur 2008; Shvachko et al. 2010). Second, it has a data processing package called Hadoop MapReduce framework. Here, the modules are loosely connected. They can either share the same set of computing nodes or be installed on different nodes. Hadoop is composed of several different daemons/servers as depicted in [Figure 11.3](#) and listed below:

- a. Name Node, Data Node, and Secondary Name Node for managing HDFS
- b. Job Tracker and Task Tracker for performing MapReduce jobs

HDFS supports master/slave architecture. The Name Node and Data Node are software packages intended to execute on commodity nodes. HDFS is implemented in Java language. Machines that support Java can execute the Name Node or the Data Node software. In an HDFS cluster, only one machine runs the Name Node module. This machine with Name Node software is called as a master. The two main functions of Name Node are management of namespace in the file system and regulating clients' access to files. All

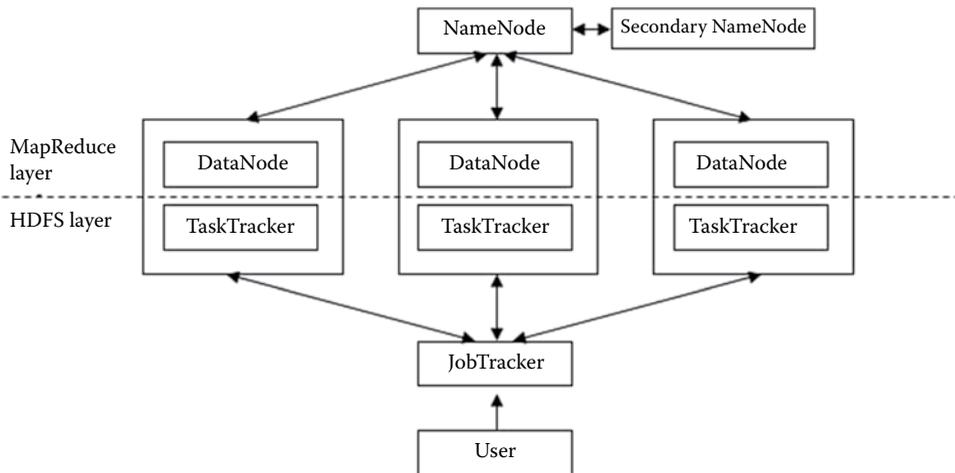


FIGURE 11.3

Hadoop architecture. (From Doukeridis, C. and K. Nørnvåg. *The VLDB Journal, Springer Link*, Vol. 23, (3), 2014: 355–380.)

HDFS metadata is stored on Name Node. The design of the system never allows user data to flow through the Name Node.

A single instance of Data Node software executes on other machines in the Hadoop cluster. These machines running Data Node software are called workers. The user data imported into HDFS are stored in files. A standard file is divided into fixed-size chunks ranging from 16 to 64 MB per block and is configurable. The Name Node is responsible for distributing these blocks to Data Nodes. Data Nodes are responsible for storing and managing the chunks assigned to them. Furthermore, the Name Node with metadata is responsible for mapping of file to block, tracking of location (Data Node) of block, and monitoring the status of Data Nodes. It performs operations on file system namespace. These operations include opening a file, closing a file, renaming of files and directories, etc. Data Nodes are responsible for handling the file system's read and write requests. According to the instructions provided by Name Node, Data Nodes support block creation, deletion, and replication. There is the provision for Secondary Name Node, which can take over tasks of Name Node if it fails at some point of time. The metadata from Name Node is copied to Secondary Name Node periodically. Thus, HDFS (Shafer et al. 2010; Agarwal et al. 2011) is a block-structured file system controlled by a single master that runs a Name Node module. It represents a simplified architecture because only one Name Node is present in a cluster.

The MapReduce engine of Hadoop consists of one JobTracker designated as master and many TaskTrackers designated as workers (slaves). JobTracker is the contact point for the user to submit its job. JobTracker partitions the input data into equal-size chunks to be processed by independent map and reduce modules. It is also accountable for fair scheduling of MapReduce jobs, assigning tasks to TaskTrackers for execution, monitoring the progress of TaskTrackers, and finally reporting and invoking the user function once all jobs are completed. On each node, the TaskTracker runs MR jobs and periodically informs the JobTracker about job completion and requests new jobs. When a new task arrives at TaskTracker, by default, a new instance of Java Virtual Machine (JVM) is spawned to run it.

11.2.3 RHadoop Platform

MapReduce is a powerful programming framework to work with a massive amount of data stored in the HDFS. R is the popular programming language mainly used for data analytics, statistical computing, and data visualization. It is highly extensible and has object-oriented features. Thus, R and Hadoop together are a natural choice to perform Big Data analytics and visualizations. Hence, RHadoop is a project developed at Revolution Analytics that provides R programmers a powerful tool to analyze the data stored in the HDFS.

RHadoop is a collection of R packages for connecting R to Hadoop and running R on Hadoop nodes. This allows users to manage and analyze data with Hadoop. RHadoop includes the following packages:

• <code>rhdfts</code>	This package provides basic connectivity to the HDFS. R programmers can browse, read, write, and modify files stored in HDFS from within R.
• <code>rmr</code>	A package that allows the R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster.
• <code>rhbase</code>	This package provides basic connectivity to the HBASE distributed database, using the Thrift server. R programmers can browse, read, write, and modify tables stored in HBASE from within R.
• <code>plyrmr</code>	This package enables the R user to perform common data manipulation operations on very large datasets stored on Hadoop. It provides a familiar plyr-like interface while hiding many of the MapReduce details.

11.3 Improvisations of K-Means Clustering Algorithm

Clustering is the most popular unsupervised machine learning (ML) technique that supports in the formation of disjoint clusters. Clustering algorithms have evident peculiar features. First, the clustering process is iterative in nature. Second, the number of iterations required to converge a clustering process depends on different factors. In some cases, it is based on the movement of cluster centroids across iterations and the algorithm converges once the centroids become stable. In some other situations, convergence is not possible even after a huge number of iterations. These cases can be handled by stating the maximum number of iterations. Finally, the clustering process may be executed in multidimensional space. Except the initial iteration, the advancement of the clustering process is based on the computations performed and the movement of cluster centroids in previous iterations. It is almost infeasible to execute expensive clustering calculations on single computing device as the number of data points increases. The major obstacle in the realization of clustering algorithm is the formation of suboptimal clusters as the algorithm is unable to find global optima. Making an effort to determine the optimal solution is even more challenging in a distributed environment where data points are partitioned on several nodes.

In this direction, (Arthur and Vassilvitskii 2007) presented a K-means++ algorithm. This algorithm selects only the first center at random from the given dataset. The subsequent centers were chosen with a probability such that they are placed far away from the previously chosen centroids. This algorithm uses the idea that good clustering is relatively spread out and so it selects new centers in a way that are far away from earlier selected centers. But K-means++ is inherently a serial algorithm whose performance degrades with

an increase in dataset size. Thus, a parallel K-means++ algorithm is presented in Bahmani et al. (2012) to deal with larger datasets in a more efficient way.

The main objective of this section is to explore and analyze the MapReduce implementations of serial K-means++ and parallel K-means++ algorithms using RHadoop framework. These algorithms are modeled as a series of three MapReduce jobs. The job of the first MapReduce is to decide on initial centroids. The second MapReduce module performs the iterative clustering step until centroids become stable between iterations. The job of the third MapReduce job is to assign data points to the nearest final centroids. Both the algorithms are compared and evaluated using two different datasets based on clustering quality, size-up analysis, total time taken for execution, and the number of iterations required for convergence. The two datasets used for analysis are Diabetes and BOW datasets taken from the UCI machine learning repository (Blake et al. 1998).

11.3.1 K-Means++ Algorithm

The simplicity and speed of the K-means algorithm have made it the most accepted clustering technique to extract information from a given dataset. But it does not provide any accuracy guarantee. Moreover, the accuracy of the algorithm depends on the choice of initial centroids. K-means++ provides a way to choose initial centroids to improve the quality of final clusters obtained by traditional K-means algorithm. This algorithm selects only the first center uniformly at random from the dataset. The later centroids are chosen one by one in a controlled manner whose selection is based on the set of previously chosen centers. The algorithm to choose initial centers is given in Table 11.1.

11.3.2 Overview of Parallel K-Means++ Algorithm

The main drawback of serial K-means++ is that it cannot be parallelized, so Bahmani et al. (2012) proposed a parallel version of the K-means++ initialization algorithm. The intuition behind this algorithm is to sample $O(K)$ points in each iteration instead of sampling a single point as it was done in serial K-means++ algorithm. Therefore, this algorithm uses an additional input in terms of oversampling factor l , which is approximately equal to

TABLE 11.1

K-Means++ Algorithm

Input	A set of n data points is represented by X such that $X = \{x_1, x_2, x_3, \dots, x_n\}$ and the number of clusters is represented by K .
Output	Output of this module is final set of initial centroids $IC_{final} = \{C_1, C_2, \dots, C_k\}$
Procedure	<p>Determine the list of initial centers $IC_{final} \leftarrow \{\}$</p> <ol style="list-style-type: none"> 1. $C_1 \leftarrow$ single data point uniformly picked at random from X 2. While $IC_{final}.length < K$ 3. for each $x_i \in X, 1 \leq i < n$ 4. Compute $D(x_i, C)$ where D is the shortest Euclidian distance from d_i to nearest center $C_j \in IC_{final}, i \neq j$ 6. if Sample $x_i \in X$ with probability $\frac{D^2(x_i, C)}{\sum D^2(x_i, C)}$ 7. $IC_{final} \leftarrow IC_{final} \cup \{x_i\}$ 8. end for 9. end while 10. Run K-means using the set of initial centers

TABLE 11.2

Algorithm for Parallel K-Means++ Clustering

Input	A set of n data points is represented by X such that $=\{x_1, x_2, x_3, \dots, x_n\}$, the number of clusters is represented by K and the oversampling factor l
Output	Output of this module is final set initial centroids $C = \{c_1, c_2, \dots, c_k\}$
Procedure	<p>Determine the list of initial centers $C \leftarrow \{\}$</p> <ol style="list-style-type: none"> 1. $C_1 \leftarrow$ single data point uniformly picked at random from X 2. Compute $\psi = \phi_X(C)$ 3. for $O(\log \psi)$ times do 4. Compute $d(x_i, C)$ where d is the shortest Euclidian distance from x_i to set C Sample each point $x_i \in X$ independently with probability $\frac{l \cdot d^2(x_i, C)}{\phi_X(C)}$ 5. $C \leftarrow C \cup \{x_i\}$ 6. end for 7. for $x \in C$, set w_x to be the number of points in X closer to x than any other points in C 8. Re-cluster the weighted points in C into K clusters

$O(K)$. Here, the algorithm selects the first centroid uniformly at random and computes ψ as shown in step 3 in the algorithm illustrated in Table 11.2. Here, ψ is given by Equation 11.5.

$$\psi = \phi_X(C) = \sum_{x \in X} d^2(x, C) = \sum_{x \in X} (\min_{c_i \in C} x - c_i)^2 \tag{11.5}$$

where

$X = \{x_1, x_2, \dots, x_n\}$ is the input dataset containing n number of data points

$K =$ number of clusters

$C = \{c_1, c_2, \dots, c_k\}$ is the current set of centroids

$x - c_i =$ Euclidean distance between the point x and c_i

$d(x, C) = \min_{c_i \in C} x - c_i =$ minimum distance between x and c_i

The algorithm then samples each point $x \in X$ with probability $l \cdot d^2(x, C) / \phi_X(C)$ for the given set of current centroids C . These sampled points are added to the set C and the value $\phi_X(C)$ is updated in each iteration. The expected number of points in the set C is $l \log \psi$. Finally, the algorithm reclusters the $O(l \log \psi)$ weighted points into K initial centroids. The algorithm for parallel K-means++ clustering is illustrated in Table 11.2, which is easily parallelized as discussed in Section 11.4.2.

11.3.3 Research Challenges

The main goal of K-means algorithm is to determine cluster centers that can minimize the intraclass variance and maximize the interclass distance. Solving K-means problem for any arbitrary input is NP-hard. But one can use a standard approach of finding an approximate solution, which cannot find globally optimum clusters but quickly determines reasonable

solutions. It can be easily parallelized, and hence can be adopted to implement in a distributed paradigm easily. But K-means has two major shortcomings:

- Arthur and Vassilvitskii (2006) have shown that the worst-case running time of the algorithm can be exponential in input size.
- The approximate solution found by the algorithm is locally optimal, which can be far from global optimum.

The second problem of finding a globally optimal solution is addressed by the serial K-means++ algorithm. K-means++ provides the procedure of selecting initial centers based on the probability distribution of data points before performing the standard K-means algorithm. To solve the first problem, one must go for parallel K-means++ algorithm.

There is no doubt that the resultant clusters obtained by serial K-means++ algorithm are more accurate compared to the clusters obtained by the traditional randomized algorithm. But there are two major downsides of serial K-means++ algorithm as given below:

- K-means++ is a serial algorithm. The current set of chosen centers influences the choice of the next center. This hinders the scalability of the algorithm when working with large-scale datasets.
- The results obtained by K-means++ algorithm are different across several executions using the same initial conditions. As the feature dimensionality and the number of data points increase, this difference in results increases.

One way to address the second problem is to reduce the variability of the results obtained by running several instances of serial K-means++ in parallel. Then, select that instance that produces the most accurate clustering result. Running several instances of K-means++ on large datasets is not feasible. Thus, to solve both the problems, this chapter modeled parallel K-means++ algorithm as MapReduce tasks on RHadoop framework. Furthermore, the serial K-means++ and parallel K-means++ algorithms are analyzed and studied in a distributed paradigm on a variety of datasets to obtain better-quality clusters.

11.3.4 Contributions toward Proposed Work

The main objective of this chapter is to realize smart clustering algorithms such as serial and parallel K-means++ for obtaining globally optimal clusters. Thus, the main contribution of this work is to analyze the performance of serial K-means++ and parallel K-means++ algorithms on RHadoop framework. The study in this section differs from earlier studies of similar nature (Desrosiers and Karypis 2011; Koren et al. 2011; Recht et al. 2011) as follows:

- The main goal is to model the serial K-means++ and parallel K-means++ algorithms as MapReduce tasks on RHadoop platform.
- The two algorithms are compared and analyzed on the basis of size-up analysis, number of iterations taken to converge, and total running time to process a diverse set of data samples.
- PCA is used to simplify the complex data and to map the data points in higher-dimensional space to the vectors that best approximate the variance in the two directions. This will in turn help to plot the data points using a visualization tool such as scatter plot.

The experimental results obtained show that parallel K-means++ is 2.759 times faster than serial K-means++. For the sampled dataset, the parallel K-means++ is 93%–96% accurate. However, the serial K-means++ is only 85%–89% accurate for the same set of sample points. Moreover, the results of the experiments are visualized using scatter plot tool as explained in Section 11.5.3.4.

11.4 Proposed Methodology

The proposed implementation for realizing K-means clustering using serial K-means++ as an initialization method is designed as a series of two MapReduce jobs. The implementation using parallel K-means++ initialization method involves four MapReduce jobs. The methodology involves three main stages as described below and depicted in [Figure 11.4](#):

- a. *Centroid selection module*: The first step is to apply one of the centroid selection strategies on the input dataset. This step is modeled as a series of two MapReduce tasks for implementing the parallel K-means++ initialization method as described in Section 11.4.2. In case of serial K-means++ initialization, this step is implemented as a sequential algorithm working on the complete dataset. Further implementation stages are common for the distributed K-means clustering procedure that may use either the serial K-means++ or parallel K-means ++ initialization method.
- b. *Iterative clustering module*: The job of the next MapReduce job is to assign data points to the nearest centers that are selected in the previous module. Furthermore, the average of all data points present in the same cluster is computed to obtain new centers. This is an iterative step, which is iterated until centroids become stable between iterations.
- c. *Cluster assignment module*: The job of the last MapReduce job is to assign the data points to resultant clusters. This step receives the complete dataset and the path to the final set of K-means centroids from the previous step as input. Finally, the data points are allotted to the nearest final centroids to obtain resultant clusters.

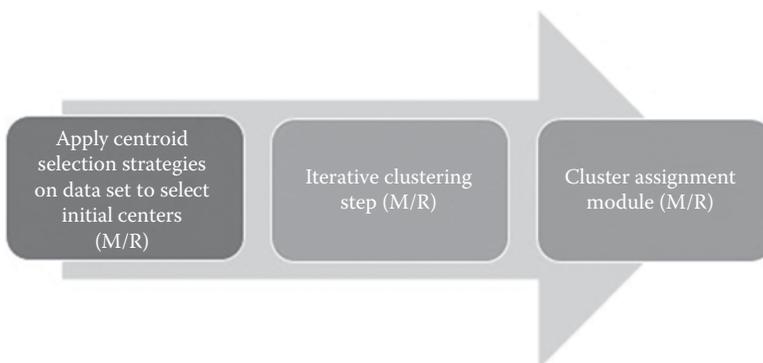


FIGURE 11.4 Methodology followed for realization of optimized clustering algorithm on RHadoop framework.

The work presented in this chapter compares the serial K-means++ and parallel K-means++ initialization methods based on accuracy, size-up analysis, and number of iterations required for convergence using two different datasets. Furthermore, the experiments are repeated for different values of $K = 30, 50,$ and 100 . One can conclude from results that parallel K-means++ is almost 2 times faster and more accurate as compared to serial K-means++.

11.4.1 Initial Centroid Selection for Serial K-Means++

This module is responsible for selecting initial centroids using serial K-means++ procedure. The input given to this algorithm is the complete dataset and the K value representing the number of clusters. This algorithm chooses the first initial centroid uniformly at random and subsequent centroids are chosen in a smart way. This is a sequential algorithm as illustrated in Table 11.1. The intuition behind this algorithm is that the good clustering is relatively spread out and selection of new centroids should be done in such a way that these (new centroids) should be far away from the previously selected centroids.

11.4.2 Initial Centroid Selection for Parallel K-Means++

Parallel K-means++ initialization method is modeled as a series of two MapReduce jobs. This algorithm uses an oversampling factor l , which is equal to the $O(K)$ (number of clusters in K-means++), and samples each point with probability $l \cdot d^2(x, C) / \phi_X(C)$ to select l centroids in each iteration as illustrated in Table 11.2. Step 2 of parallel K-means++ explained in Table 11.2 is modeled as a first MapReduce job, where the mapper function independently work on each input partition $X_i \subseteq X$, where $1 \leq i \leq s$ and s represents the number of partitions to compute $\phi_{X_i}(C)$, and the reducer can simply add the values obtained from all mappers to compute $\phi_X(C)$. Similarly, step 4 of the parallel K-means++ can be modeled as a second MapReduce job where each mapper samples independently to obtain l/s initial centroids. The job of the reducer is to combine these initial centroids obtained from all partitions to get l initial centroids, which is approximately equal to $O(K)$ as explained in Bahmani et al. (2012).

- *MapReduce Module for Computing the Cost Function:* The objective of this module is to compute the cost function $\phi_X(C)$, where $\phi_X(C)$ represents the cost of X with respect to C and is given by Equation 11.1. Initially, the input data are split into equal-size chunks and each mapper takes the dataset split as input and computes the $\phi_{X_i}(C)$ as illustrated in Table 11.3. The job of the reducer is to just add these $\phi_{X_i}(C)$ to obtain $\phi_X(C)$ as illustrated in Table 11.4.
- *Map Function for Selecting Initial Centroids:* This map function is to receive the dataset split X_i such that $X_i \subseteq X$ and sample each point in the subset with the probability $l \cdot d^2(x_{ij}, c_k) / \phi_{X_i}(C)$ to obtain a list C_i of l/s centroids such that $C_i = \{c_1, c_2, \dots, c_{l/s}\}$, selected for that particular subset X_i . The algorithm for the map function is illustrated in Table 11.5.
- *Reduce Function for Selecting Initial Centroids:* As illustrated by the algorithm in Table 11.6, the job of the reducer is to just combine the input received by all mappers to obtain l initial centroids.

11.4.3 Design of Iterative Clustering Module on RHadoop

This module is common for both serial K-means++ and parallel K-means++ algorithms. This is the next MapReduce job whose responsibility is to iteratively generate cluster

TABLE 11.3

Map Module for Computing the Cost Function $\phi_X(C)$

Input	The input is the dataset partitioned into equal-size chunks. Let $X_i \subseteq X$ be dataset split such that $X_i = \{x_{i1}, x_{i2}, x_{i3}, \dots, x_{in_i}\}$ containing $n_s \equiv n/s$ number of data points of dimensionality m , where $x_{ij} \in X_i$ represents a data point, $1 \leq i < s$, s represents number of partitions, n_s represents the number of data points in X_i , and n is the size of the complete dataset. The value K representing the number of clusters and the set C representing the current set of centroids is given as input.
Output	The output is a list of key/value pairs consisting of $(1, \phi_{X_i}(C))$, where $\phi_{X_i}(C)$ represents a cost function for each dataset split X_i .
Algorithm	<ol style="list-style-type: none"> 1. for each $x_{ij} \in X_i, 1 \leq j \leq n_s, 1 \leq i \leq s$ 2. Compute the $\phi_{X_i}(C) = \sum_{x_{ij} \in X_i} d^2(x_{ij}, C) = \sum_{x_{ij} \in X_i} (\min_{c_k \in C} x_{ij} - c_k)^2$

TABLE 11.4

Reduce Module for Computing the Cost Function $\phi_X(C)$

Input	A pair of $(1, \{\phi_{X_1}(C), \phi_{X_2}(C), \dots, \phi_{X_s}(C)\})$, where key is 1 and value the list of cost functions for each dataset split computed by respective map function.
Output	The output the final value of $\phi_X(C)$ computed for the complete dataset X .
Algorithm	$\phi_X(C) = \phi_{X_1}(C) + \phi_{X_2}(C) + \dots + \phi_{X_s}(C)$

TABLE 11.5

Map Module for Selecting l/s Initial Centroids

Input	The input is the dataset partitioned into equal-size chunks. Let X_i be the dataset split such that $X_i = \{x_{i1}, x_{i2}, x_{i3}, \dots, x_{in_i}\}$ containing $n_s \equiv n/s$ number of data points of dimensionality m , where x_{ij} represents a data point, $1 \leq i \leq s$, s represents the number of partitions, n_s represents the number of data points in X_i , and n is the size of the complete dataset. The value of $\phi_X(C)$ is computed in the previous step. The value K representing the number of clusters, the updated set C representing the current set of centroids, and the oversampling factor $l = \theta(K)$ is given as input.
Output	The output is a list of l/s centroids $C_i = \{c_1, c_2, \dots, c_{l/s}\}$ selected for subset X_i .
Algorithm	<ol style="list-style-type: none"> 1. Let $C_i \leftarrow \{\}$ 2. for each $x_{ij} \in X_i, 1 \leq j \leq n_s, 1 \leq i \leq s$ 3. Compute $d(x_{ij}, C)$ where d is the shortest Euclidian distance from x_{ij} to set C 4. Choose any random value r between $[0, 1]$ 5. Sample $x_{ij} \in X_i$ with probability $p_{d_i} = \frac{ld^2(x_{ij}, C)}{\phi_X(C)}$, 6. if $(p_{d_i} > r)$ $C_i \leftarrow C_i \cup x_{ij}$ 7. end for each

TABLE 11.6

Reduce Module for Selecting $l = O(K)$ Initial Centroids

Input	The input is a list C_i of l/s centroids such that $C_i = \{c_1, c_2, \dots, c_{l/s}\}$ selected for each dataset split X_i from all the mappers and the updated set C representing the current set of centroids.
Output	The output is the final set of l initial centroids $C = \{c_1, c_2, \dots, c_l\}$. for the complete dataset X .
Algorithm	$C \leftarrow C \cup \{C_1\} \cup \{C_2\} \cup \dots \cup \{C_s\}$

TABLE 11.7**Map Function for Iterative Clustering Module**

Input	The initial cluster centroids from the previous MapReduce module, which are represented by set $C = \{c_1, c_2, c_3, c_4, \dots, c_K\}$, where K is the number of clusters and the dataset split $D_s = \{d_0, d_1, d_2, \dots, d_{n_s-1}\}$ containing $n_s \leq n$ data points of dimensionality m and d_j represents a data point, where $0 \leq j < n_s$, and n is the size of the complete dataset.
Output	The points are designated to the clusters. The output is the list of (c_i, d_j) pairs, where c_i is the centroid-id and d_j represents the data point assigned to a particular centroid-id c_i , where $0 \leq i < K$ and $0 \leq j < n_s$.
Algorithm	<ol style="list-style-type: none"> 1. $minDistance \leftarrow$ Maximum value that can be held by $minDistance$ 2. for each $c_i \in C, d_j \in D_s, 0 \leq i < K, 0 \leq j < n_s$ 3. $distance \leftarrow \sqrt{\sum_{x=0}^{m-1} (c_i - d_j)^2}$ 4. if $distance < minDistance$ 5. $minDistance \leftarrow distance$ 6. $assignedCentroid \leftarrow c_i$ 7. end if 8. end for 9. Output $(assignedCentroid, d_j)$

centroids using initially selected cluster centroids from the previous step and the dataset split. The role of the reducer is to traverse the output received from the mapper and compute the new arithmetic mean for data points that represents a new updated centroid point.

- *Map Function:* This map function is responsible for assigning data points to the nearest cluster centroids received from the previous section. The output of this module is the list of (c_i, d_j) pairs, where c_i is the centroid-id and d_j represents the data point assigned to a particular centroid-id c_i , where $0 \leq i < K$ and $0 \leq j < n_s$. This map module is depicted in [Table 11.7](#).
- *Reduce Function:* The reducer function receives a list of K key/value pairs of the form $(c_i, \text{list of data points in this cluster})$, where $0 \leq i < k$. This list is traversed and the arithmetic mean is used to compute new cluster centroids; this is described in [Table 11.8](#).

11.4.4 Cluster Assignment Module on RHadoop

This function reads the file containing final cluster centroids produced in the previous step and uses the dataset split to find the closest cluster centroid to each data point. It outputs pairs of cluster centroid and data point to represent these assignments. The design for this module is depicted in [Table 11.9](#).

11.5 Results and Discussions

K-means algorithm has become one of the most sought-after data analytical techniques. Recent times have seen a lot of research carried out to further improve this popular

TABLE 11.8

Reduce Function for Iterative Clustering Module

Input	Pairs of the form $(c_i, \{d_0, d_1, d_2, \dots, d_{j-1}\})$, where the key is a cluster centroid, where $0 \leq i < K$ and value is a list of data points in this cluster where $j \geq 1$.
Output	A list of pairs (l, c_j) , where $0 \leq i < K$, which contains the new K-means centroids produced.
Algorithm	<ol style="list-style-type: none"> 1. for each pair $(c_i, \{d_0, d_1, d_2, \dots, d_{j-1}\})$, $0 \leq i < K$, $j \geq 1$ 2. $newCentroid \leftarrow \sum_{x=0}^{j-1} d_x / j$ [Use of Arithmetic Mean] 3. Output $(l, newCentroid)$ 4. end for

algorithm. This section presents a comparative review of serial K-means++ and parallel K-means++ on MapReduce paradigm using two different datasets. The main focus of this section is to show how initial centroid selection strategies help to improve the accuracy and the performance of conventional K-means algorithm. Furthermore, these initialization methods are evaluated on size-up analysis of how well the system adapts to the increase in dataset size by keeping the number of computing nodes constant. Here, the experiments are conducted using a single pseudo-node cluster. Moreover, the experiments are repeated for different values of $K = 30, 50$, and 100 on two datasets of different sizes. Next, the final clusters obtained are evaluated using the sum of squared error (SSE) criterion. This section also presents the use of the PCA technique to map the dataset to lower dimensions in order to plot the data points by utilizing a visualization tool called scatter plots. Thus, this section describes the datasets used and the experimental setup, and presents a discussion on analyzing centroid selection strategies based on accuracy, size-up analysis, and number of iterations required for convergence.

TABLE 11.9

Map Function for Cluster Assignment Module

Input	The updated set of cluster centers from the previous module represented by $C = \{c_1, c_2, c_3, \dots, c_K\}$, where K is the number of clusters. let Y be the set of 'n' data points and d_j represents a data point such that each $d_j \in Y$
Output	A list of pairs (c_i, d_j) where $0 \leq i < K$ and $0 \leq j < n$, which represents the cluster assignment pairs produced.
Procedure	<ol style="list-style-type: none"> 1. $clusterCentroids \leftarrow \{c_1, c_2, \dots, c_K\}$ 2. for each $d_j \in Y$, $0 \leq j < n$ 3. $minDistance \leftarrow$ Maximum value that can be held by $minDistance$ 4. for each $c_i \in clusterCentroids$, $1 \leq i \leq K$ 5. $distance \leftarrow \sqrt{\sum_{l=0}^{m-1} (d_{j1} - c_{i1})^2}$ 6. if $distance < minDistance$ 7. $minDistance \leftarrow distance$ 8. $pos \leftarrow i$ 9. end if 10. end for 11. Output (c_{pos}, d_j) 12. end for

11.5.1 Dataset Used

Two different datasets are used to evaluate various algorithms discussed in the previous section. They are Diabetes and BOW datasets. Both these datasets are downloaded from the UCI machine learning repository. The first dataset explored is Diabetes dataset, which is sampled to obtain around 1000 data points. The experimental work clusters this dataset using nine features. These features are Regular Insulin Dose, NPH Insulin Dose, Ultra Lente Insulin Dose, Unspecified Blood Glucose Measurement, Pre-Breakfast Measure, Post-Breakfast Measure, Pre-Lunch Measure, Post-Lunch Measure, and count of Hyperglycemia Symptoms. This higher-dimensional space is mapped to two-dimensional space vectors using PCA to simplify the data and help to visualize them easily. This will in turn help to interpret the data. The second dataset is BOW dataset consisting of around 4 million data points. The data points are three dimensional in space. This dataset contains five text collections in the form of BOW. For each text collection, D is the number of documents, W is the number of words in the vocabulary, N is the total number of words in the collection, and NNZ is the number of nonzero counts in the BOW. For each text collection, two files are provided, namely, `docword*.txt` (the BOW file in sparse format) and `vocab*.txt` (the vocab file). The format of the `docword*.txt` file is three header lines, followed by NNZ triples:

```

---
D
W
NNZ
docID wordID count
docID wordID count
docID wordID count
docID wordID count
...

```

The format of the `vocab*.txt` file is a line containing `wordID=n`. These datasets are ideal for clustering and topic modeling experiments.

11.5.2 Experimental Setup

This section presents the experimental setup used to carry out the clustering task on Hadoop framework. The algorithms are designed as a series of two MapReduce jobs executed on a single-node RHadoop cluster. RHadoop provides an R wrapper on top of Hadoop. The node on which RHadoop is installed has a dual-core AMD 2.5 GHz processor, 8 GB RAM, 1 TB hard drive, and Ubuntu 12.04 LTS server OS. All five daemons required to carry out MapReduce jobs run on a single-node Hadoop cluster. The five daemons are NameNode, JobTracker, Secondary NameNode, DataNode, and TaskTracker. Apache Hadoop 1.2.1 is used and source codes are compiled using Revolution R enterprise 7.4.1 server.

11.5.3 Performance Evaluation

This section discusses different metrics to evaluate the performance of the serial K-means++ and parallel K-means++ algorithm on RHadoop platform using datasets of different sizes. The evaluation of the algorithms is based on size-up analysis, number of iterations required

for convergence of the algorithms, evaluating cluster validation, determining optimal number of clusters, and finally visualizing the results obtained using scatter plot tool.

11.5.3.1 Size-Up Analysis for Distributed Serial K-Means++ and Parallel K-Means++ Algorithm

Size-up analysis is used to check the performance of the system with increase in size of the dataset by keeping the number of nodes constant. This section presents the experimental results for size-up analysis using pseudo-node Hadoop cluster in the R environment. The evaluation of the algorithms based on size-up analysis is depicted in [Figure 11.5](#).

The graph shows very good size-up analysis for distributed parallel and serial K-means++. The running time for both the algorithms is increasing linearly as the number of data points increases. It also depicts that the parallel K-means++ performs better than serial K-means++ algorithm.

11.5.3.2 Total Time Taken to Converge by the Algorithms and Number of Iterations Required for Convergence

This section compares and analyzes distributed clustering algorithms on the basis of the total time taken to converge. Here, the total time taken for convergence of algorithms is equal to the system time (processor time) and user time (time taken to execute the program). The system time is computed using the `system.time()` API in R language. Experiments are conducted on two datasets, namely, Diabetes and BOW datasets consisting of around 1000 and 4 million data points, respectively. Experimental results depicts that the total time taken to obtain final clusters using parallel K-means++ is relatively much less than the time taken by serial K-means++ algorithm. This is because parallel K-means++ is selecting the centers more judiciously compared to serial K-means++. This in turn makes K-means parallel much faster than its counterpart without sacrificing the quality of clustering. The results obtained representing the system time and user time for two different datasets are recorded in [Table 11.10](#). The same is depicted in [Figures 11.6](#) and [11.7](#) for Diabetes and BOW datasets, respectively.

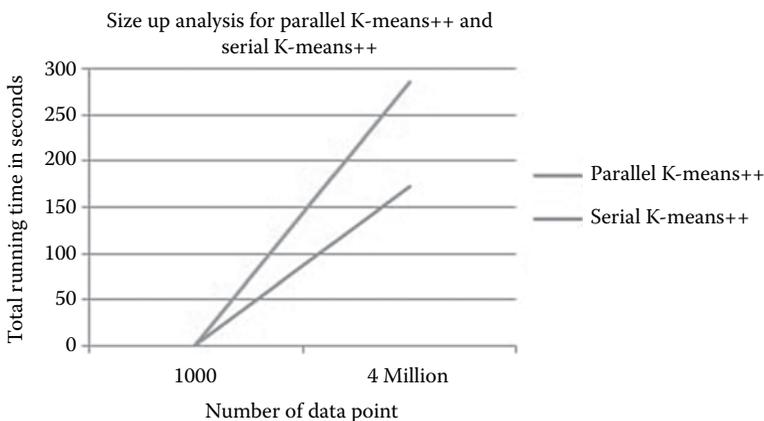


FIGURE 11.5

Size-up analysis for distributed serial K-means++ and parallel K-means++ on RHadoop.

TABLE 11.10

Comparing the Total Time Taken to Obtain Final Clusters

Dataset	K	Serial K-Means++ (Time in s)		Parallel K-Means++ (Time in s)	
		User Time	System Time	User Time	System Time
Diabetes 1000 samples	30	0.036	0.022	0.025	0.020
	50	0.033	0.018	0.030	0.017
	100	0.038	0.020	0.033	0.018
BOW 4 million samples	30	33.3	46.11	10.11	20.12
	50	36.29	136.43	13.15	98.23
	100	40.23	250.23	44.18	115.15

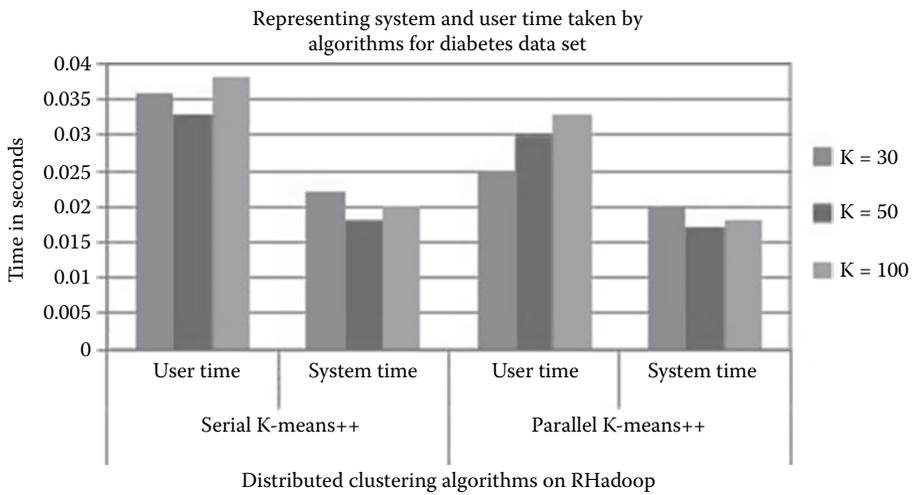


FIGURE 11.6

Comparing the system and user time taken by distributed algorithms for Diabetes dataset.

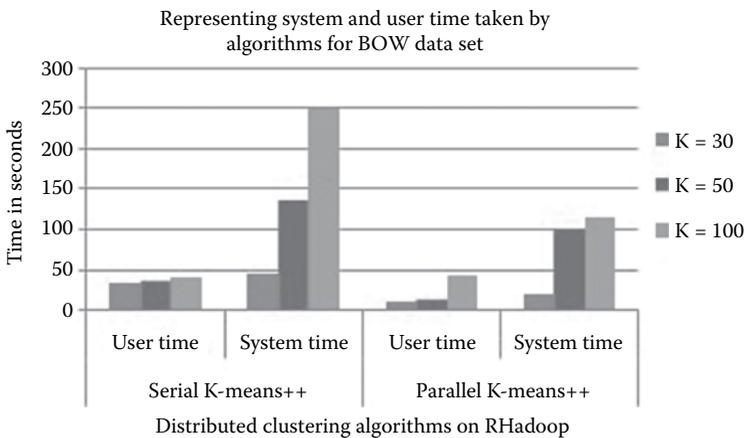


FIGURE 11.7

Comparing the system and user time taken by distributed algorithms for BOW dataset.

TABLE 11.11

Comparing the Number of Iterations Taken to Converge by Distributed Clustering Algorithms

Dataset	K	Number of Iterations Required to Converge	
		Serial K-Means++	Parallel K-Means++
Diabetes 1000 samples	30	5	4
	50	8	5
	100	20	8
BOW 4 million samples	30	40	24
	50	60	56
	100	100	72

In addition to this, Table 11.11 shows the number of iterations to converge by the two algorithms. This experiment is repeated for different values of K. The results show that the parallel K-means++ typically requires less number of iterations to converge as compared to serial K-means++ algorithm.

The results achieved clearly indicate that parallel K-means++ is more efficient and a better algorithm compared to serial K-means++. For smaller sampled datasets, the execution time of both the algorithms is almost the same. But with larger datasets such as BOW, parallel K-means++ is around 2.759 times faster than serial K-means++ as shown below:

$$\frac{\text{Execution time of serial K-means++}}{\text{Execution time of parallel K-means++}} = 2.75969 \tag{11.6}$$

Thus, this section compared and evaluated the parallel and serial K-means++ based on total time taken to obtain final clusters and number of iterations required to converge.

11.5.3.3 Evaluations of Clusters and Determining Optimal Number of Clusters

Many numerical measures are applied to judge cluster validations, which are classified into three types: external index, internal index, and relative index. Here, the experimental work makes use of internal index measure such as SSE criteria to evaluate clustering quality without any help of external information (ground truth values). It is a measure of discrepancy between the data and an estimation model. A small SSE indicates a tight fit of the model to the data. For each data point, the error is the distance to the nearest cluster. To get SSE, we square these errors and sum them, as given below:

$$SSE = \sum_{i=1}^k \sum_{x \in C_i} (x - m_i)^2 \tag{11.7}$$

where x is a data point that belongs to C_i , m_i is the centroid point of cluster C_i , and K is the number of clusters.

Here, SSE is also utilized to compute the optimal number of clusters for Diabetes dataset, which is sampled to consist of around 1000 data points. Applying SSE to determine the optimal number of clusters is called elbow method. Here, first, SSE is computed for different values of K (e.g., $K = 2, 4, 6, 8$ etc.) as shown in Figure 11.8. As described

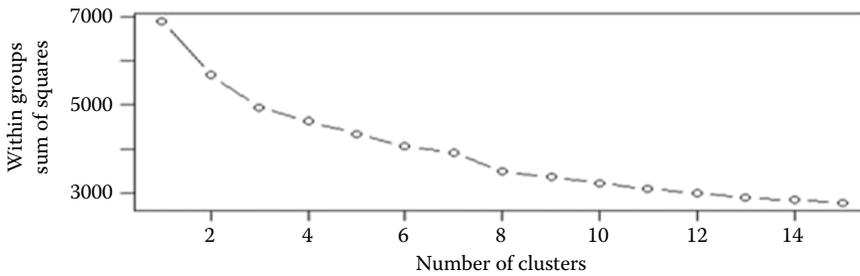


FIGURE 11.8
Optimal number of clusters for sampled Diabetes dataset.

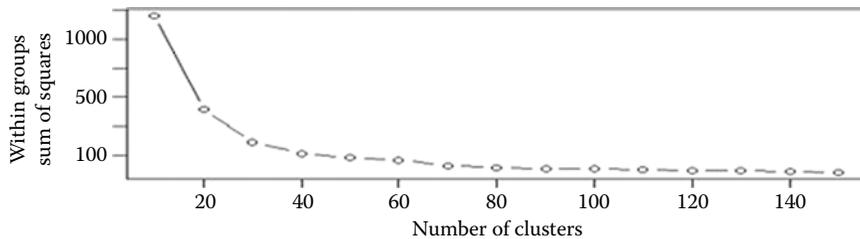


FIGURE 11.9
Optimal number of clusters for BOW dataset.

above, the SSE is defined as the sum of the squared distance between each member of the cluster and its centroid. When the value of K is plotted against SSE, then one can observe a decrease in error as the value of K increases. This is because as the number of clusters decreases, the size of clusters becomes smaller. Hence, the distortion is also lesser. The idea behind the elbow method is to choose the value of K at which the SSE value decreases steeply. In [Figure 11.8](#), $K = 8$ is the value the elbow method has selected. Similarly, in [Figure 11.9](#), according to the elbow method, $K = 40$ is the appropriate number of clusters for BOW dataset. This is a heuristic method; hence, it may or may not work for a particular case.

The experimental results show that the sampled dataset with parallel K-means++ gives an accuracy of 93%–96%. On the contrary, the same dataset with serial K-means++ achieves an accuracy of 85%–89%. This clearly indicates that parallel K-means++ is more efficient and a better algorithm compared to serial K-means++.

11.5.3.4 Visualization Results

This section presents the visualization results obtained for Diabetes and BOW datasets. These results are obtained for serial K-means++ algorithm. [Figures 11.10](#) and [11.11](#) show the visualization results for Diabetes dataset with $K = 16$ and BOW dataset for $K = 100$, respectively.

Scatter plot tool is used to plot these visualization results. These visualizations will help to interpret the data more appropriately and facilitate to know a few facts about the underlying data. [Figure 11.11](#) presents a more dense plot for $K = 100$ clusters and consists of around 4 million data points.

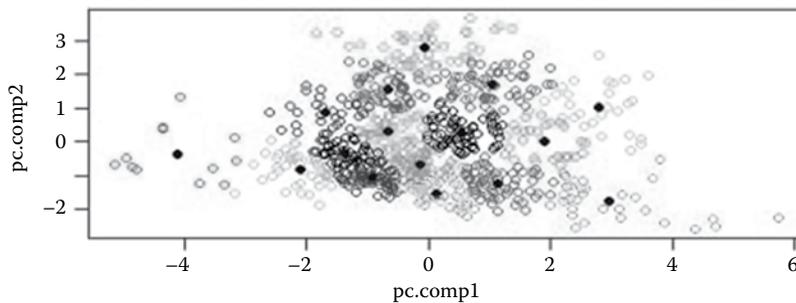


FIGURE 11.10
Visualization plot showing 16 clusters ($k = 16$) for Diabetes dataset.

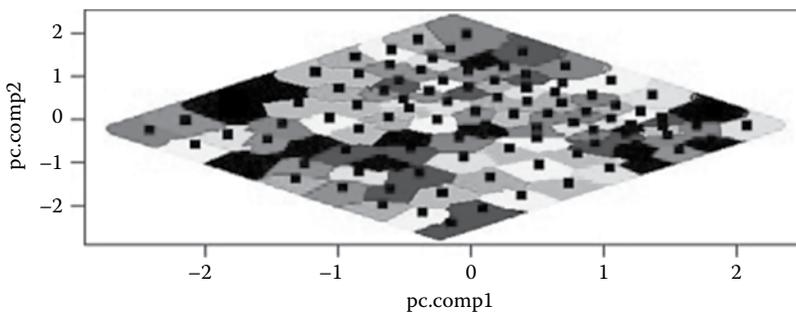


FIGURE 11.11
Visualization plot showing 100 clusters ($K = 100$) for Bag of Words dataset.

References

- Agarwal, S., D. Borthakur, and I. Stoica. *Snapshots in Hadoop Distributed File System*. UC Berkeley Technical Report UCB/EECS, 2011.
- Arthur, D. and S. Vassilvitskii. *How Slow Is the K-Means Method?* ACM, New York, USA, pp. 144–153, 2006.
- Arthur, D. and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, Louisiana, pp. 1027–1035. Society for Industrial and Applied Mathematics, 2007.
- Bahmani, B., Moseley, B. Vattani, A. Kumar, R. and S. Vassilvitskii. Scalable K-means++. *ACM Proceedings of the VLDB Endowment*, Vol. 5, Article no. (7), 2012: 622–633.
- Blake, C., E. Keogh, and C.J. Merz. UCI Repository of Machine Learning Databases. Department of Information and Computer Science, Irvine, 1998.
- Borthakur, D. HDFS Architecture Guide. *Hadoop Apache Project*, 2008: 53.
- Dean, J. and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, Vol. 51(1), 2008: 107–113.
- Desrosiers, C. and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor (eds), Springer, USA, pp. 107–144, 2011.
- Doulkeridis, C. and K. Nørnvåg. A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal, Springer Link*, Vol. 23, (3), 2014: 355–380.

- Grolinger, K., M. Hayes, W. Higashino, A. L'Heureux, D. S. Allison, and M. A. M. Capretz. Challenges for MapReduce in Big Data. In *Proceedings of the IEEE 10th 2014 World Congress on Services (SERVICES 2014)*, June 27–July 2, 2014, Alaska, USA.
- Guo, J., Du, L. Li, Y. Zhao, G. and Jiya, J. Distributed data platform system based on Hadoop platform. In *Proceedings of International Conference on Computer Science and Information Technology*, Springer, Chennai, India, January 2014, pp. 533–539.
- Jain, A. K., M. N. Murty, and P. J. Flynn, Data clustering: A review. *ACM Computing Surveys*, Vol. 31, 1999: 264–323.
- Jiang, D., C. Tang, and A. Zhang, Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, 2004: 1370–1386.
- Hruschka, E. R. et al. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, Vol. 39.2, 2009: 133–155.
- Kanungo, T., D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and Angela Y. Wu. An efficient K-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, (7), 2002: 881–892.
- Karloff, H., S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, Austin, Texas, pp. 938–948. Society for Industrial and Applied Mathematics, 2010.
- Koren, Y. and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor (eds), Springer, USA, pp. 145–186, 2011.
- Lee, Kyong-Ha, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with MapReduce: A survey. *ACM SIGMOD Record*, Vol. 40, (4), 2012: 11–20.
- Pantel, P. A. Clustering by Committee, PhD Thesis, Department of Computer Sciences of the University of Alberta, Canada, 2003.
- QiuHong Li, Peng Wang, Wei Wang, Hao Hu, Zhongsheng Li, and Junxian Li. An efficient K-means clustering algorithm on MapReduce. *Journal of Database Systems for Advanced Applications, Springer Link*, Vol. 8421, 2014: 357–371.
- Rao, B. T. and L. S. S. Reddy. Survey on improved scheduling in Hadoop MapReduce in cloud environments. *arXiv preprint arXiv:1207.0780*, 2012. e-print service operated by Cornell University.
- Recht, B., C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor et al. (eds), Granada, Spain, pp. 693–701, 2011.
- Rokach, L. and O. Maimon (eds). Clustering methods. *Data Mining and Knowledge Discovery Handbook*. Springer, USA, pp. 321–352, 2005.
- Shafer, J., S. Rixner, and A. L. Cox. The Hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, New York, USA, pp. 122–133. IEEE, 2010.
- Shvachko, K., H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, New York, USA, pp. 1–10. IEEE, 2010.
- Xu, R. and D. Wunsch II, Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, Vol. 16, 2005: 645–678.
- Yoon, Min, Hyeong-il Kim, Dong Hoon Choi, Heeseung Jo, and Jae-woo Chang. Performance analysis of MapReduce-based distributed systems for iterative data processing applications. *Mobile, Ubiquitous, and Intelligent Computing*, Vol. 274, Springer, 2014: 293–299.

12

Security and Privacy: Challenges and Defending Solutions for NoSQL Data Stores

Sandhya Aneja and Nagender Aneja

CONTENTS

12.1 Security Solutions of Traditional Database Management Systems	237
12.1.1 Role-Based Access Control for Database Systems.....	238
12.1.2 Case Study: Role-Based Access Control for PostgreSQL	239
12.1.3 Case Study: PostgreSQL User Authentication.....	240
12.2 NoSQL Vulnerabilities Based on System Model	241
12.2.1 System Model of NoSQL Data Stores.....	241
12.2.2 Case Study: Role-Based Access Control for MongoDB	241
12.2.2.1 Creating User and Authentication.....	241
12.2.2.2 Role-Based Access Control Model in MongoDB	242
12.2.3 Information Extraction Threat	244
12.2.4 SQL Injection Attack.....	244
12.3 Security and Privacy Solutions for NoSQL Data Stores.....	244
12.3.1 Temporal Role-Based Access Control Model.....	244
12.3.2 Spatial and Location-Based Access Model.....	245
12.3.3 Content-Based Access Control	245
12.3.4 Fine-Grained Access Control	246
12.3.5 Attributes-Based Access Control	246
12.3.6 Context-Aware RBAC Model.....	246
12.3.7 Role and Attributes-Based Access Model	247
12.4 Privacy.....	248
12.4.1 Privacy Violation under NoSQL Data Stores	248
12.4.2 Privacy Preserving Solutions	248
12.4.3 Conclusion	249
References.....	249

12.1 Security Solutions of Traditional Database Management Systems

Relational database management systems (RDBMSs) have traditionally been used to store and manage data from Internet, Intranet, or Desktop applications in order to serve multiusers systems. However, there are security flaws in network services, web browsers, operating systems, and database systems. This section discusses security of traditional database systems using an example of PostgreSQL database system (The PostgreSQL Global Development Group n.d.-b; Bernier 2009). Role-based access control (RBAC) frameworks

(Sandhu 1996; Sandhu et al. 1996, 2000; Ahn and Sandhu 2000) have been proposed to manage access control which use basic idea of administrative roles and policies to decide authorization of any entity over any resource. This section explains RBAC and its variations with an example of PostgreSQL.

12.1.1 Role-Based Access Control for Database Systems

In traditional RDBMSs, RBAC models have been implemented in commercial products like Oracle, MySQL, and PostgreSQL and many more with some variations from each other. A set of Users "U," Roles "R," Permissions "P," and Sessions "S" are maintained in RBAC models. First RBAC model called Flat RBAC, as shown in Figure 12.1, creates roles for users from set "U" in accordance with administrative roles from set "R" in organizations and allows permissions based on privileges from set of permissions "P." Roles own different types of objects like tables, schema, views, etc. Different users and groups of users may be assigned different roles, and as per consequence, that user or group of users own objects under that role. Relation between roles and users is many-to-many, that is, any user may have any number of roles and a role may be assigned to many users. Similarly, roles-to-permissions relationship is also many-to-many. Flat RBAC model uses principle of least privilege. According to the principle of least privilege, permissions that are actually required to complete a task by members are the only permissions assigned to the role.

Second RBAC model, as shown in Figure 12.2, includes inheritance that is based on role hierarchies in organization while creating roles. Every senior role may be inherited from its junior role. This can be in two possible forms where senior role may either take an arbitrary number of permissions or may take a limited number of permissions. For example, tasks that are work in progress may be private for junior role to prohibit senior role to access work-in-progress tasks. Therefore, some objects under junior role may not be permitted under inheritance.

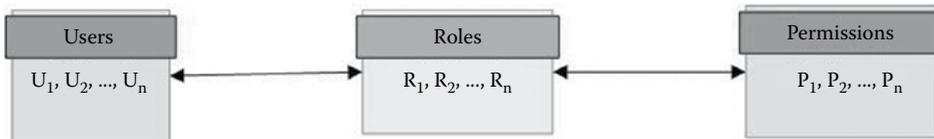


FIGURE 12.1 Flat RBAC.

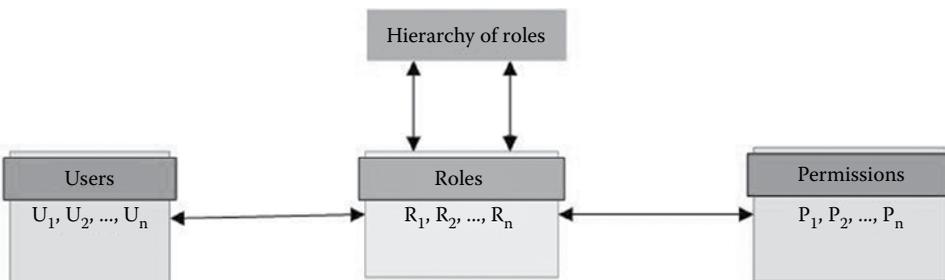


FIGURE 12.2 RBAC model with inheritance.

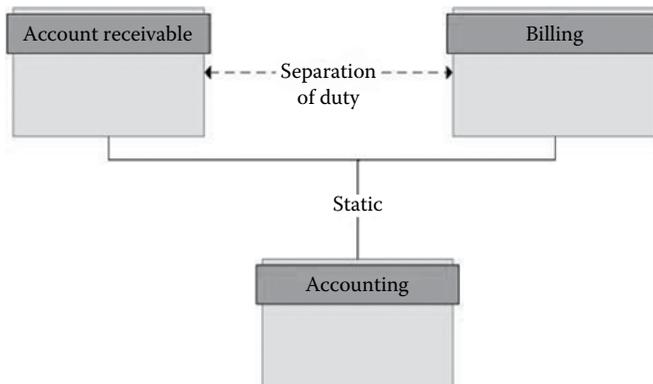


FIGURE 12.3
Constrained RBAC.

Third model called constrained RBAC, as shown in [Figure 12.3](#), includes separation of duties (SOD) which suggests that mutually exclusive roles must be invoked to complete a sensitive task. For example, as shown in [Figure 12.3](#), Billing Supervisor and Account Receiving Officer role must be disjoint as billing supervisor need not to know regarding total amount with the department. SOD may be static and dynamic. Under dynamic SOD, sessions are used to decide the set of permissions for each role participating in each session.

In the fourth model called symmetric model, a set of permissions and a set of objects which a user and a group of users under assigned role are reviewed. These sets of permissions and objects may be assigned directly or indirectly based on the inheritance requirements. Review may be required when multiple roles are required to work together and when any user leaves or is promoted in the organization. Data abstraction is supported by all the models wherein it is required to include objects and permissions according to setup of organization.

12.1.2 Case Study: Role-Based Access Control for PostgreSQL

PostgreSQL manages database access permissions using concept of roles. In PostgreSQL, a role is considered as a user or as a group of users which may be created either using “create role” command or using “create user” command. “Create user” command considers login by default, while using “create role” command, other parameters like password, type of user, and other privileges can also be set. First role which is created by default is “postgres,” and it is created with encrypted password and with privileges like create databases, tables, columns, rows, views, new roles, and others (The PostgreSQL Global Development Group n.d.-b). Privileges that are used by “CREATE ROLE” command are INHERIT, CREATEDB, CREATEROLE, REPLICATION, and SUPERUSER. In case a Role is created with INHERIT privilege, it can take other roles using SET ROLE. A required set of permissions may be created as group role and assigned to other roles. For example, for web-based applications, a group role web is created and different users as per roles are included as members. [Figure 12.4](#) shows a hierarchy of roles in setup of organization which is maintained using INHERIT and SET ROLE commands.

Certain challenges that a DBA should consider while creating roles and their controls to prevent a malicious or an unauthorized user accessing resources are explained by Robert Bernier (Bernier 2009). For example, any user logged in with its credentials may access “pg

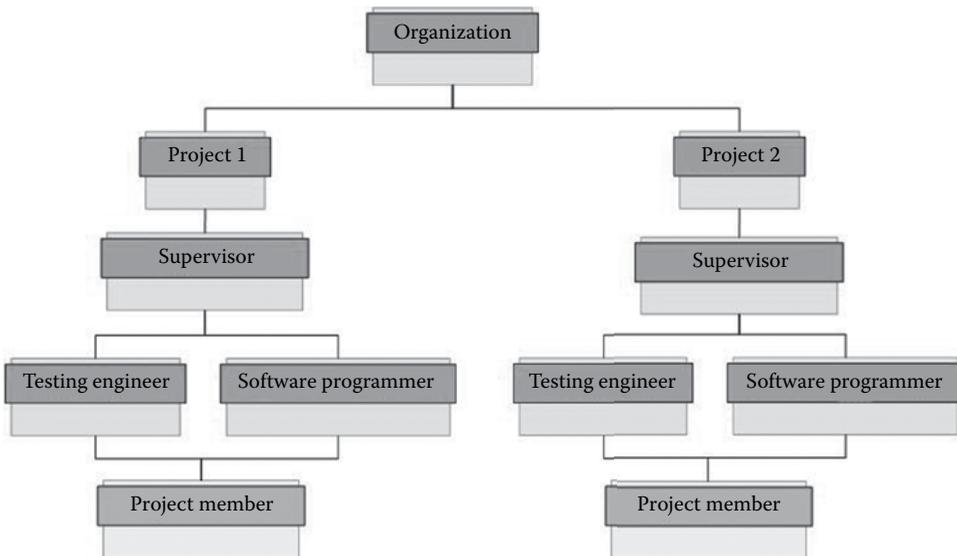


FIGURE 12.4
Setup of organization.

user table (a system table)" under public schema and glean information about all the users under cluster. PostgreSQL (The PostgreSQL Global Development Group n.d.-a) implements row-level security that restricts rows to be returned when queries are executed over database to fetch or modify data into the database. PostgreSQL provides row-level security with the feature of writing policies. Policies may be specified for permission to access explicit rows since by default no rows may be modified and accessed. Row-level security is fine level of access control which PostgreSQL provides to users to secure the objects like table. PostgreSQL also provides column-level security using views and its access to different users.

12.1.3 Case Study: PostgreSQL User Authentication

Accessibility of data and resources over database server is maintained using RBAC mechanism; however, before access, a client application first needs to connect to database server wherein it is checked that if a particular application is authorized to connect the database server with the specified user name, host address, and database name. Client application needs to submit credentials of corresponding role. It may be encrypted or unencrypted. Encrypted passwords are stored using MD5Hash as per the policy mentioned in "pg_hba.conf" file. PostgreSQL may be configured using Kerberos, Lightweight Directory Access Protocol (LDAP), encryption external protocols for encrypted user credentials, and Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocol for authenticated communication.

```
postgres# psql -U "projectSupervision" -P "password"
```

Most of the RDBMSs, as presented for PostgreSQL, provide RBAC with fine-grained access control (FGAC). RDBMS has also been known to provide flexible services with a wide range of scalability. However, beyond a range of scalability (Chandra 2015), RDBMS

performance degrades due to its complex expensive operations to fetch and modify data and high-level interfaces to access and use. This scalability failure and lack of a clear picture of consequences due to data growth have led more and more developers toward NoSQL options. NoSQL database technology offers flexibility, scalability, and performance that present day web-scale application developers demand. Thus, traditional RDBMSs are secure but not as flexible, scalable, and efficient as NoSQL data stores. Furthermore, NoSQL data stores have features required for big data applications but still not secure. Next section explains system model of NoSQL and its vulnerabilities with case study of MongoDB.

12.2 NoSQL Vulnerabilities Based on System Model

12.2.1 System Model of NoSQL Data Stores

System model adopted for NoSQL data stores is schema-less, scalable, and flexible. Schema-less means application dumps data without any structure in the database. Due to being schema-less, it is faster and supports newly extended nonuniform types at any point of time. But, while querying the data from data stores, a structure is required to retrieve information; thus, an implicit structure is required or is maintained within documents of data stores. Implicit structure is generally imposed while designing the application and assumed as a general structure for databases.

Scalability in data stores means data stores scales with high transparency to application when volume of data increases even when data are distributed over multiple servers. NoSQL data stores include sharding to distribute data automatically according to hardware availability and which shows same performance with increase in available hardware. NoSQL data stores are flexible in terms of deployment over multiple servers at one region or lying over globally different regions over cloud (Shermin 2013). With all these challenges to meet over RDBMS, NoSQL data stores development overlooks security concerns. Nevertheless, like RDBMS, basic RBAC model is provided in NoSQL data stores. In next section, we discuss about the basic RBAC model in context of MongoDB.

12.2.2 Case Study: Role-Based Access Control for MongoDB

In this section, we explain the procedure to create user, roles, and functionalities provided in MongoDB for user authentication and access control. However, in MongoDB, the basic model used for implementation of RBAC is the same as PostgreSQL. PostgreSQL also provides fine level of security called row-level security, which is difficult to implement in MongoDB since its model is document-oriented.

12.2.2.1 Creating User and Authentication

In MongoDB (MongoDB, n.d.-a), “`db.createUser()`” command is used to create a user. The method takes username and password. In addition, any other information may be stored in the same document with third argument `customData`. Method returns an error if username already exists. Default database in MongoDB is “`admin`.”

```
>db.createUser({ user: "Admin", pwd: "NoSQL", customData: {Id: 1}})
>Successfully added user
```

TABLE 12.1

Comparison of MongoDB and PostgreSQL on Authentication Protocols

	Authentication		
	Default Authentication Protocol	SSL, Kerberos, LDAP External Protocol Support	ClusterMode and Internal Authentication
PostgreSQL	Md5 hash function	Yes	No
MongoDB	SHA-1 hash function	Yes	Yes, with Sharding and MapReduce

Methods to manage users for database are `db.removeUser()`, `db.dropUser()`, and `db.getUser()`. Different methods to grant and revoke permissions over documents of database are used to update users. By default, user authentication is not enabled in MongoDB. When a client tries to connect through MongoDB, the client is first authenticated and then it can get access. We use following commands to start the MongoDB server:

```
>mongod -port 27017 -auth -dbpath D:/data/db
>mongo -u Admin -p NoSQL
>use admin
>switched to db admin
```

Like PostgreSQL, MongoDB also supports secure communications using SSL/TLS, and Kerberos or LDAP external party protocols-based authentication. Also, it supports X.509 certificate-based authentication. Users need to submit certificate rather than username and password. Authentication may be activated over each system as part of cluster (MongoDB, n.d.-c).

MongoDB supports SSL/TLS authentication at both standalone and cluster-based server systems (MongoDB, n.d.-c). For replicaset mode, cluster nodes and master node passwords are stored in configuration file. Nonetheless, it is possible for attacker to overpass the authentication process (Okman et al. 2011). [Table 12.1](#) presents comparison on authentication protocols used in PostgreSQL and MongoDB. Internal authentication is also used in cluster mode while replicating and sharding data among the nodes.

12.2.2.2 Role-Based Access Control Model in MongoDB

Roles may be assigned to users using `createUser()` or `createRole()` methods. Major roles used in MongoDB are: `read`, `readWrite`, `dbAdmin`, `userAdmin`, `clusterAdmin`, `readAnyDatabase`, `read-WriteAnyDatabase`, `userAdminAnyDatabase`, and `dbAdminAnyDatabase`. Using, “`db.createRole()`” method, any number of roles with different types of privileges on resources may be created as follows:

```
use admin db.createRole(
{
  role: "myClusterwideAdmin",
  privileges: [
    { resource: { cluster: true }, actions: [ "addShard" ] },
    { resource: { db: "config", collection: "" }, actions:
[ "find", "update", "insert", "remove" ] },
    { resource: { db: "users", collection: "usersCollection" }, actions:
[ "update", "insert", "remove" ] },
```

```

    { resource: { db: "", collection: "" }, actions: [ "find" ] }
  ],
  roles: [
    { role: "read", db: "admin" }
  ]
},
)

```

Two roles are created in above commands. The first role can use “find” operation on all collections of database “config” and sharding in cluster, etc., and the second role “role” is created on admin database. A role created in admin database can inherit roles in any database. A role including multiple roles inherits all the privileges of the included roles. A role can inherit privileges from other roles in database. For example, myClusterwideAdmin Role may be granted to projectSuperVisior Role using grantRolesToRole command. For a user, if more privileges are required, it may be set by creating new roles for particular user using db.grantRolesToUser() method. db.revokeRolesFromUser() method can be used to revoke certain privileges (MongoDB, n.d.-b).

```

db.runCommand({grantRolesToRole: "ProjectSuperVisior", roles:
["myClusterwideAdmin"]})

```

Basic RBAC model, discussed in the last section for MongoDB, is not sufficient. MongoDB is based on document-oriented data model wherein documents are stored in a collection in hierarchical manner. When query is run to fetch data, present RBAC model either grants access or denies to the whole collection. In other words, it means accessibility at more fine-grain-level like field based, which neither can be implemented nor can be enforced (Colombo and Ferrari 2015a). For example, in a collection for organization of e-mails, e-mail as one document and e-mail content as one of the fields, any user with access to this collection would see all the contents of e-mails of other employees. Another important feature which is not available in current NoSQL data stores is the support for attributes-based access control (ABACs), temporal-, location-, and context-based access controls as explained below. This is a key requirement since different applications may need different access control for NoSQL. We believe that context, attributes, and fine-grain-level related information should also be used for access control.

In spite of support for standalone as well as cluster-wide system, internal and external authentication, following subsections present vulnerabilities of NoSQL data stores. [Table 12.2](#) presents comparison of the authentication protocols used in PostgreSQL and MongoDB.

TABLE 12.2

Comparison of MongoDB and PostgreSQL on Access Control

Comparison on Access Control					
	Principle of Least Privilege	Inheritance	Static SOD	Dynamic SOD	Row- and Column-Level Security
PostgreSQL	Yes	Yes	No, one role active at a time	Yes, using sessions	Yes, using update, insert, delete
MongoDB	Yes	Yes	Yes, multiple roles at a time	Yes, using sessions	No

12.2.3 Information Extraction Threat

In NoSQL data stores, data files are unencrypted and any attacker having access to file system may extract the information from files. One possible solution is to store the information in data stores in encrypted form. If authentication is not activated, attacker can extract information. In case authentication is activated, Kerberos-like high-level protocols are needed to restrict the attacker. SQL injection attack is another information extraction threat that is explained below.

12.2.4 SQL Injection Attack

User authentication can be handled using cryptographic solutions (Aviv et al. 2015). Secure Shell (SSL) protocol and other third-party solutions in Aviv et al. (2015) are proposed to activate and achieve user authentication and secure internode communication between client and server. Furthermore, some of the vulnerabilities of NoSQL data store make them prone to insider or external attacks like “Injection Attacks.”

MongoDB supports JSON representation for documents to store data in data store. However, this suffers from “Injection Attack” (Aviv et al. 2015). Injection attack is a scenario wherein an attacker can login to a system without having username and password and can also get information of all users that exists in the system. In addition, in NoSQL data store, a sharded mode is run by MapReduce methods, and these methods use java scripts. These java scripts take input from user which again in case of malicious user can be altered and an intended MapReduce method may be changed by malicious user. A malicious user may run a script on data store by manipulating the input parameters at interface. The solution to handle this attack is to disable the execution of java scripts over data store. Another vulnerability of NoSQL data stores is to expose NoSQL data stores to HTTP REST API (Aviv et al. 2015) using which an application submits its query to data store. A malicious user can control over data store using this vulnerability and extract the all intended information.

12.3 Security and Privacy Solutions for NoSQL Data Stores

For security of NoSQL data stores, different cryptographic techniques and access control techniques may be used. The authors in Colombo and Ferrari (2015a), Hu et al. (2015), Bertino et al. (2014), Colombo and Ferrari (2015b), and Li et al. (2010) proposed fine-grained context-aware access control mechanisms including spatial and location-based information. In Hu et al. (2013), ABAC is used which does not use RBAC as basic access control. Implementation of context-aware access control is based on platform and data models of NoSQL big data tools, for example, key-value pairs at attributes and row levels (Ali et al. 2015), etc. could be further extended. This section explains possible modifications proposed for basic RBAC.

12.3.1 Temporal Role-Based Access Control Model

Temporal role-based access control (TRBAC) emphasizes that some roles may be time dependent and available to users at certain time periods. For example, there may be two

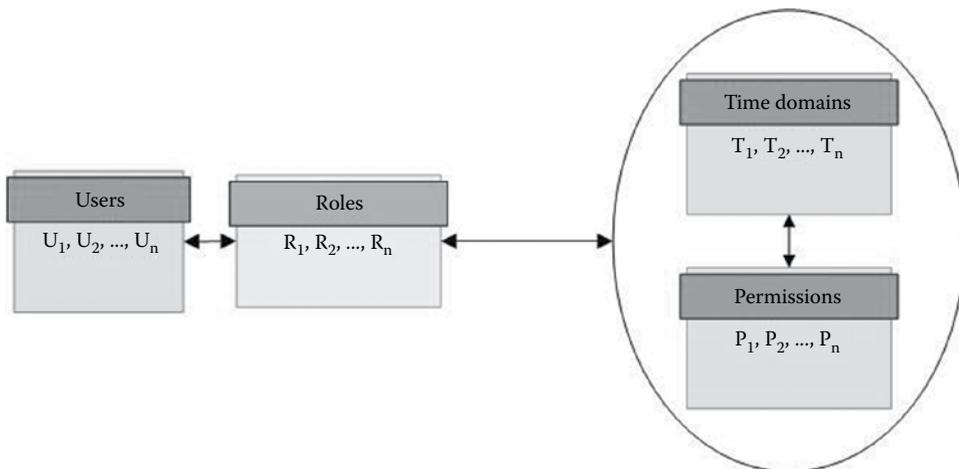


FIGURE 12.5
Temporal role-based access model.

shifts, daytime and nighttime, and two different teams for both the shifts. Roles associated with both the shifts may be same or different to some extent. In TRBAC, there is a time-based field which takes values from a set of time intervals. TRBAC (Bertino et al. 2001) can be enabled periodically as per user's expectation using role triggers wherein role triggers may be prioritized based on conflicting actions. In TRBAC, as shown in Figure 12.5, permissions with respective roles are activated with time domain. TRBAC is useful in the scenarios where staff works under multiple departments. Depending on the time domain (Liu et al. 2016), users may take permissions under different departments.

12.3.2 Spatial and Location-Based Access Model

Spatial and location-aware RBAC (Geo-RBAC) model is an enhanced RBAC model with location information of user who is trying to access data store. Geo-RBAC is based on the fact that a role should be allowed to operate only if its location is within permissible spatial domain. A set of possible spatial domains with specified roles, permissions, and operations are maintained. A user under spatial roles is permitted only under corresponding permissions. A logical location (Bertino et al. 2005) may be used instead of a real one by using a logical mapping function. For example, in a health service organization, a doctor is permitted to perform certain operations, but a user in a doctor role is only allowed performing operations in premises of the organization. Permissions are associated with spatial roles, and the user under spatial role inherits all the permissions associated with roles and also can have some specific permissions. Figure 12.6 shows a core Geo-RBAC model.

12.3.3 Content-Based Access Control

It is required in some applications to grant or deny access to data depending on its content. For example, audios and videos in surveillance applications have to be protected from security as well as privacy perspective (Bertino et al. 2001).

Multimedia data collected under Smart Home applications for child and elder cares also require content-based access control rather than only RBAC.

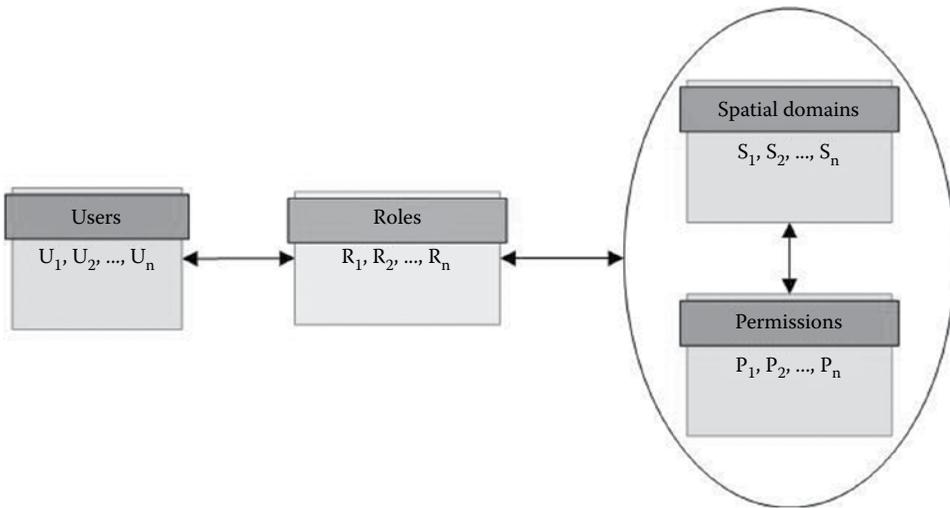


FIGURE 12.6
Geo-RBAC model.

12.3.4 Fine-Grained Access Control

RBAC suffers a limitation because according to RBAC, permissions are associated with roles which means that instances of a role has the same set of permissions (Li et al. 2010). For example, in health service organization in case of two roles doctor and patient, if as per policy doctor role has permission to update and read patients records, then all doctors can see records of all patients. However, a doctor should be allowed to see records of only his or her own patients. This limitation may be overcome with FGAC policy which states that roles and permissions should be maintained as an index set. Corresponding to each user, an index role is assigned, and corresponding to indexed role, permissions are assigned.

12.3.5 Attributes-Based Access Control

Attributes-based access control (ABAC) model (Hu et al. 2015) suggests to use attributes of subject (person or device) and attributes of object (resource) to write policies and to write a set of different permissions over different resources to grant access control to subjects over objects. In RBAC, main driving factor is "Role of Subject" which is used to decide the access control, whereas in ABAC, "Subject and Objects" decides. Attributes of subjects and objects may be time, location, content, environmental conditions like temperature, and context. Policies are a set of conditions for deciding access controls. A set of relations combine object and subject attributes. Boolean operations over a combined set of relations and conditions using computational logic finally implement access control. [Figure 12.7](#) depicts the components of ABAC model.

12.3.6 Context-Aware RBAC Model

Context-aware role-based access model (CA-RBAC) (Kulkarni and Tripathi 2008) also modifies RBAC by keeping a set of users or roles and a set of permissions of RBAC model and including a set of contexts say C . Context set includes all main dynamic attributes like time, location, content, and status. Now, as per CA-RBAC, RTC is the set of conditional

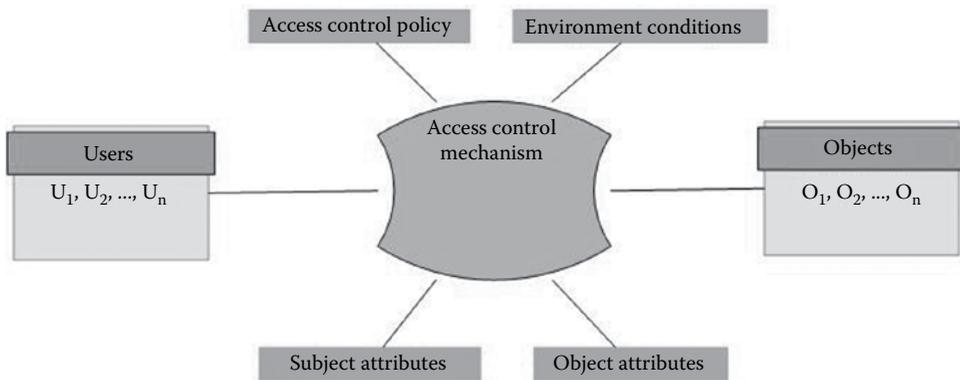


FIGURE 12.7
Attributes-based access control.

expressions which evaluates true or false for a given role from the set of roles and context from the set of contexts. Now, for a requested access control as per different roles, permissions, and context, conditional expression in RTC evaluates true; it is granted, otherwise it is denied.

12.3.7 Role and Attributes-Based Access Model

Objects and subjects combined together may have n attributes and while dynamically deciding the policies based on 2^n different combinations is a difficult task in real scenario. Also, there may be n static roles, and it may be easier to give 2^n different permissions in static scenario. However, there is need to have dynamic ABAC as well. Thus, from n required permissions where say n_1 are static and n_2 are dynamic, a system can have $2^{n_1} + 2^{n_2}$ permissions which are much lesser than 2^n . Role and attribute-based access model (R-ABAC) simplifies complexity involved in implementation of ABAC, quickly decides a set of permissions, overcomes risk involved in deciding control in run time, and overcomes the lack of dynamic feature in RBAC (Kuhn et al. 2010).

Table 12.3 presents comparison of RDBMSs and NoSQL data stores on possible different access control models. R-ABAC model (Kuhn et al. 2010) seems to be a possible solution to provide fine-grain-level access control. But when the amount of data is huge, the implementation of R-ABAC with required level of efficiency may be a challenge to meet.

TABLE 12.3
Comparison of RDBMSs and NoSQL Data Stores on Implementation of Access Controls

Comparison on Different Access Controls			
	TRABC/Geo-RBAC/ Content-RABC/CA-RBAC	ABAC	R-ABAC
RDBMS	Using triggers on schema-oriented DBs (Bertino et al. 2001, 2005)	Using column and row-based privileges	Using triggers, row- and column-level privileges
NoSQL	Difficult for schema-less DBs	Difficult because of size of data (Kuhn et al. 2010)	Size of data will be reduced (Kuhn et al. 2010)

12.4 Privacy

Privacy is an important factor for data stores in addition to the security. Privacy is concerned with accessibility of information selectively. It is still required to gain insight into whether there is breach of privacy with data sharing. Privacy is data dependent, and thus privacy solution may vary from one kind of application to another. For a specific application, there is lot more scope to improve data sharing, data trustworthiness, and data provenance (Bertino 2012; Khan and Mane 2013). Context may be stored based on document levels, database levels, collection level, or field level. Contexts may include location, temporal and other quantifiable information. While implementing access control (Cybersecurity 2014) on different contexts and on the basis of user request, it is interesting to study and compare the quantity and quality of data that are filtered during access control. However, time overhead involved for such filtering while enforcement of control may be high. Also, there are varieties of NoSQL data stores with different query languages and data models (Shermin 2013). Thus, it seems ambitious to develop a general privacy preserving mechanism which include all possible contexts and may be applicable to different data-dependent applications. In this section, we explain how privacy may be violated in NoSQL stores. We also show how privacy preserving techniques are provided while using, maintaining, and sharing data across NoSQL stores.

12.4.1 Privacy Violation under NoSQL Data Stores

Data are collected in data stores for extracting useful and relevant information. Data may be stored for business and security purposes, or they may be stored by governments for keeping records of their citizens. Data are analyzed for extracting information, and for this purpose, data may be shared among different agencies. Following are the scenarios where privacy may be violated while maintaining data records:

1. *Data matching*. Consider a scenario where a suspicious passenger is to be searched from a list of passengers who traveled on a specific day. This searching requires data from different sources to be matched which may violate privacy requirements (Bertino et al. 2014).
2. *Biometric authentication*. Usually, biometric information is used for identification. Government and other private agencies maintain biometric data of users. Biometric data are sensitive information. Privacy should be maintained while transferring data over the systems used for identification (Bertino et al. 2014).
3. *Collaborative mining*. Agencies may share data to do collaborative mining which may violet, in general, privacy of users who shared their information (Bertino et al. 2014).

12.4.2 Privacy Preserving Solutions

A directed acyclic graph may be maintained while sharing the data, where each node represents an entity which accessed data. Operations on data may be labeled on edges. These graphs may be used to see that privacy is preserved while analyzing and sharing data. Data should also be kept in encrypted form. All operations like analyzing, mining, and matching should be processed in encrypted form. Machine learning techniques may be used to evolve different policies to evolve solutions which preserve privacy.

12.4.3 Conclusion

Success of Cassandra for Facebook, Google BigTable for Google, Dynamo for Amazon, MongoDB for Ebay, and many other NoSQL tools have shown great performance for practical scenarios where data are continuously growing. Data are important in all applications, and thus there is need to extend authentication and authorization as proposed solutions to handle vulnerabilities. However, system models and size of data used for NoSQL data stores are still challenges for research community.

References

- Ahn, G.-J. and Sandhu, R. 2000. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4), 207–226. Retrieved from <http://doi.acm.org/10.1145/382912.382913>. DOI: 10.1145/382912.382913.
- Ali, S., Rauf, A., and Ahmad, J. 2015. Protecting unauthorized big data analysis using attribute relationships. *The Science International (Lahore)*, 27(6), 5075–5077.
- Aviv, R., Alexandra S., and Emanuel, B. 2015. NoSQL, No Injection? Examining NoSQL Security. arXiv preprint arXiv:1506.04082.
- Bernier, R. 2009. Total Security in a PostgreSQL Database. Retrieved from <https://www.ibm.com/developerworks/library/os-postgresecurity/os-postgresecurity-pdf.pdf>
- Bertino, E. 2012. Data protection from insider threats. *Synthesis Lectures on Data Management*, 4(4), 1–91.
- Bertino, E., Bonatti, P. A., and Ferrari, E. 2001. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3), 191–233. Retrieved from <http://doi.acm.org/10.1145/501978.501979>. DOI: 10.1145/501978.501979.
- Bertino, E., Catania, B., Damiani, M. L., and Perlasca, P. 2005. GEO-RBAC: A spatially aware RBAC. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, Stockholm, Sweden, June 1–3, pp. 29–37.
- Bertino, E., Ghinita, G., Kantarcioglu, M., Nguyen, D., Park, J., Sandhu, R., Sultana, S., Thuraisingham, 2014. A roadmap for privacy-enhanced secure data provenance. *Journal of Intelligent Information Systems*, 43(3), 481–501.
- Chandra, D. G. 2015. Base analysis of NoSQL database. *Future Generation Computer Systems*, 52, 13–21.
- Colombo, P. and Ferrari, E. 2015a. Complementing mongoDB with advanced access control features: Concepts and research challenges. In *23rd Italian Symposium on Advanced Database Systems, SEBD 2015*, Gaeta, Italy, June 14–17, 2015, pp. 343–350.
- Colombo, P. and Ferrari, E. 2015b. Privacy aware access control for Big Data: A research roadmap. *Big Data Research*, 2(4), 145–154.
- Fidelis, Cybersecurity. 2014. Current Data Security Issues of NoSQL Databases. Retrieved June 2016, from <http://docplayer.net/8000242-Current-data-security-issues-of-nosql-databases.html>
- Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M. Schnitzer, A. et al. 2013. Guide to Attribute Based Access Control (ABAC) Definition and Considerations (Draft). NIST Special Publication, 800(162).
- Hu, V. C., Kuhn, D. R., and Ferraiolo, D. F. 2015. Attribute-based access control. *Computer*, 48(2), 85–88.
- Khan, S. and Mane, V. 2013. SQL support over MongoDB using metadata. *International Journal of Scientific and Research Publications*, 3(10), 1–5.
- Kuhn, D. R., Coyne, E. J., and Weil, T. R. 2010. Adding attributes to role-based access control. *IEEE Computer*, 43(6), 79–81.
- Kulkarni, D. and Tripathi, A. 2008. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT'08)*, Estes Park, Colorado, pp. 113–122.

- Li, J., Zhao, G., Chen, X., Xie, D., Rong, C., Li, W., Tang, L., and Tang, Y. 2010. Fine-grained data access control systems with user accountability in cloud computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (cloudcom)*, Indianapolis, USA, November 30–December 3, pp. 89–96.
- Liu, C., Peng, Z., and Wu, L. 2016. Role of time-domain based access control model. *Journal of Software Engineering and Applications*, 9(02), 57.
- MongoDB. n.d.-a. Access Control Tutorials. Retrieved June 2016, from <https://docs.mongodb.com/manual/administration/security-checklist/>
- MongoDB. n.d.-b. Administration Roles. Retrieved June 2016, from <https://docs.mongodb.com/manual/reference/built-in-roles/#database-administration-roles>
- MongoDB. n.d.-c. Authentication Tutorials. Retrieved June 2016, from https://docs.mongodb.com/manual/core/authentication/&_ga=1.190704814.486653156.1464747095#x-509-certificate-authentication
- Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., and Abramov, J. 2011, November. Security issues in NoSQL databases. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, IEEE, Changsha, China, pp. 541–547. DOI: 10.1109/TrustCom.2011.70.
- Sandhu, R. 1996. Issues in RBAC. In *Proceedings of the First Acm Workshop on Role-Based Access Control*, New York, New York: ACM, pp. 47–63. Retrieved from <http://doi.acm.org/10.1145/270152.270162>. DOI: 10.1145/270152.270162.
- Sandhu, R., Ferraiolo, D., and Kuhn, R. 2000. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control*. New York, New York: ACM. Retrieved from <http://doi.acm.org/10.1145/344287.344301>. DOI: 10.1145/344287.344301.
- Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E. 1996, February. Role-based access control models. *Computer*, 29(2), 38–47.
- Shermin, M. 2013. An access control model for NoSQL databases (Unpublished doctoral dissertation). The University of Western Ontario, Ontario, Canada.
- The PostgreSQL Global Development Group. n.d.-a. Create Policy. Retrieved June 2016, from <https://www.postgresql.org/docs/9.5/static/sql-createpolicy.html>
- The PostgreSQL Global Development Group. n.d.-b. PostgreSQL Documentation. Retrieved June 2016, from <https://www.postgresql.org/docs/9.5/static/user-manag.html>

13

Challenges and Security Issues of Distributed Databases

Neha Gupta and Rashmi Agrawal

CONTENTS

13.1 Introduction to Security in Distributed Databases.....	251
13.2 Fundamentals of NoSQL Security	252
13.2.1 Reasons for Security Threats in Various NoSQL Databases	253
13.3 Comparison of Relational Database Security and NoSQL Security	255
13.4 Security Threats with NoSQL Databases.....	256
13.4.1 Distributed Environment	256
13.4.2 Authentication.....	257
13.4.3 Safeguarding Integrity.....	258
13.4.4 Fine-Grained Authorization and Access Control	258
13.4.5 Protection of Data at Rest and in Motion	258
13.4.6 Privacy of User Data.....	260
13.5 NoSQL Security Reference Architecture.....	260
13.5.1 NoSQL Cluster Security.....	260
13.5.1.1 User Access Management: Authentication.....	261
13.5.1.2 Authorization.....	261
13.5.1.3 Auditing	262
13.5.1.4 Encryption.....	262
13.5.1.5 Environmental and Process Control	262
13.5.2 Data-Centric Approach to Security.....	262
13.6 Security Weaknesses of Cassandra and MongoDB	263
13.7 Securing NoSQL Applications.....	265
13.8 Understanding Security Weakness in a Typical NoSQL Database: MongoDB	266
13.8.1 Advantages of MongoDB over RDBMS	267
13.8.2 MongoDB Security Architecture	268
13.8.3 Security Measures in MongoDB.....	269
References.....	269

13.1 Introduction to Security in Distributed Databases

A distributed database is logically a single database in which subdatabases are physically located at more than one place and each database is connected via the network. A distributed (Tabrez, 2013) database management system (DDBMS) is a system which provides the

management of a distributed database and marks the distribution transparent to the users. Major components in a distributed database are as follows:

1. Distributed query processor
2. Distributed transaction manager
3. Distributed metadata manager
4. Distributed integrity manager
5. Distributed security manager

Key advantages of spending distributed databases are their resiliency, high performance, scaling, reduced network traffic, and reduced network cost. However, distributed database systems are more complex to make undisputable data, and furthermore, these data also need to be carefully partitioned to make the system efficient. Above all, security is the main concern in distributed database, and it is much more challenging to maintain security in distributed databases as compared to central database. Distributed database considers all security apprehensions of a single-location database along with several additional security concerns. For a distributed database, the most important question to answer for security is where to grant system access, for which two strategies may be framed: granting system access to users at their home location and granting system access at their remote location. The major security concerns are multiple entry points, exchange of encryption keys, and effect of corrupted node on rest of the system. In multilevel, secure, distributed database architecture, several nodes are interconnected by a multilevel secure network and thus ensure system security. Recently, in few years, security impact in distributed database tools has become an emerging technology. These technologies include data mining and data warehousing systems, distributed object systems, and collaborative computing systems. For example, in a data warehouse, it is not straightforward task to integrate the security policies of different data sources, and systems need extension to function in a distributed environment.

13.2 Fundamentals of NoSQL Security

NoSQL databases have been designed to deal with large volumes of unstructured data and to give real-time performance. NoSQL databases are nonrelational distributed databases. NoSQL databases support massive data storage across multiple storage clusters. Because of the growing data and the infrastructure needs, majority of the web 2.0 companies are adopting NoSQL databases. However, security is the major concern for NoSQL databases. Most of the NoSQL databases do not provide embedded security features in the database itself, and developers need to entail security in the middleware. NoSQL systems are, by and large, a new generation of databases that focus on scale-out issues first and use the application layer to implement security features. The very interesting fact about the NoSQL database is that no two NoSQL database solutions are same since they are designed to meet the specific requirements of particular cloud-based applications (Deka, 2014).

Relational databases (Mohamad and Obay, 2014) have adopted highly secure mechanisms to provide the security services, but they also face many security threats like SQL injections, cross-site scripting, root kits, etc. NoSQL has inherited the security issues of

traditional RDBMSs along with its own due to the enhanced features of the NoSQL databases. NoSQL databases are always prone to security attacks due to the unstructured nature of data and the distributed computing environment. In NoSQL database environment, the nodes are distributed to enable parallel computing that increases the attack surface, which results in providing more complexity for security. Apart from (Ahmed and Gulmeher, 2014) distributed environment, data from a variety of nodes move from one node to another, which leads to sharing of data and increases the risk of theft.

Some of the major security concerns are as follows (Factor, 2013):

1. Insufficient encryption support for the data files
2. Weak authentication between the client and servers
3. Very simple authorization without the support for role-based access control (RBAC)
4. Vulnerability to SQL injection
5. End point input validation/filtering
6. Granular access control
7. Insecure computation
8. Insecure data storage and communication
9. Privacy preserving data mining and analytics
10. Attribute relationship methodology
11. Attribute encryption

Security experts have realized the danger of relying on perimeter security as Java scripts or JSON can be used to attack the firewalls to get unauthorized access of the data from database. In addition, granular access controls required to separate roles and responsibilities of user are not provided by many systems. NoSQL database may become even more susceptible to exploits if attackers' learning curve is over and they are able to identify hidden security or software weaknesses.

Data protection and access control are some of the key issues of security in NoSQL. To prevent unauthorized access to data stores and to maintain availability of secure data, data storage and transaction logs are also devised.

Attribute relationship (Ebrahim and Mohammad, 2015) methodology is a popular method to impose security in NoSQL databases. Protecting the valuable information is the main goal of this methodology. It (Ebrahim and Mohammad, 2015) focuses on attribute relevance in big data as a key element to extract the information. In this methodology, it is assumed that the attribute with higher relevance is more important than other remaining attributes.

Another suitable data access control method for NoSQL databases is attribute-based encryption. Attribute encryption (Ebrahim and Mohammad, 2015) is a method that allows data owners to encrypt data under access policy such that only those users who have permission to access data can decrypt it.

13.2.1 Reasons for Security Threats in Various NoSQL Databases

Broadly, NoSQL databases are classified into four categories:

1. Document-based NoSQL databases
2. Key-value-based NoSQL databases

- 3. Column-oriented databases
- 4. Graph databases

Document-based NoSQL databases map a key to some document that contains structured information. These systems store documents in a JSON or JSON-like format (Deka, 2014). In column-oriented databases, a key identifies a row containing data stored in one or more column families. Key-value databases of NoSQL are scalable, highly available, distributed, and open source (Deka, 2014). A graph database is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data. A key concept of the system is the graph (or edge or relationship), which directly relates data items in the store. All the above-mentioned databases have their own security risks which are illustrated in the following table.

		Types of NoSQL Databases			
		Document-Based NoSQL Databases	Key-Value-Based NoSQL Databases	Column-Oriented Databases	Graph Databases
Reasons for threats 	Example Text	MongoDB, CouchDB All data in document-based NoSQL database are stored as plain text, and there is no encryption mechanism to encrypt data files. This means that any malicious user with access to the file system can extract the information from the files	Cassandra, Voldemort, Redis All data are stored as plain text, and there is no encryption mechanism to encrypt data files. Data encryption is possible if BerkeleyDB is used as the storage engine	HBase, HyperTable Does not support data encryption	Neo4J Does not support data encryption
	Password	The passwords are encrypted by MD5 hash algorithm, MD5 algorithm, or PBKDF2 hash algorithms which are not very secure algorithms	All passwords (Ebrahim and Mohammad, 2015) in these databases are encrypted by the use of MD5 hash function, and the passwords are very weak. Any malicious user can bypass client authorization; user can extract the data because there is no authorization mechanism in internode message exchange	Passwords are not encrypted	Passwords are not encrypted
	Communication	Uses SSL (Ebrahim and Mohammad, 2015) with X.509 certificates for secure communication between user and database cluster and intracluster authentication	Communication between client and server is not encrypted as it does not support string escaping concept	Relies on SSH for internode communication. It does not tolerate the failure of range server, and if a range server crashes, it will not be able to recover the lost data	Uses SSL for communication between client and server

Continued

Types of NoSQL Databases				
	Document-Based NoSQL Databases	Key-Value-Based NoSQL Databases	Column-Oriented Databases	Graph Databases
Potential for attack	JavaScript is used as an internal scripting language, and it has potential for scripting injection attack and denial of service attack	Prone to injection attack and has problem in managing inactive connection. It has potential for denial of service attack because it performs one thread per one client	Not vulnerable to injection attacks and denial of service attacks	Prone to injection attack
Authentication and authorization	Does not support authentication and authorization when running in shared mode	No authorization mechanism in internode message exchange	It supports user authentication by the use of simple authentication and security layer (SASL) with Kerberos. It also supports authorization by ACL	No authentication and authorization mechanism
Inline auditing	Does not support inline auditing	Does not support inline auditing	Does not support inline auditing	Does not support inline auditing

As discussed earlier, most of the NoSQL databases lack data encryption, so encrypting the sensitive database fields is needed to secure the database. Some of the databases have vulnerability for injection. It is needed to use sufficient input validation to overcome this vulnerability (Ebrahim and Mohammad, 2015). Some of them have no authentication mechanism, and some of them have weak authentication mechanism (Ebrahim and Mohammad, 2015). So, to overcome this weakness, it is needed to have strong authentication mechanism.

13.3 Comparison of Relational Database Security and NoSQL Security

The following table illustrates the differences between relational databases and NoSQL databases (Nance and Losser, 2013) on the basis of various parameters.

S. No.	Point of Difference	Relational Databases	NoSQL Databases
1	Transaction reliability	Guarantees very high transaction reliability as they fully support ACID properties	Do not guarantee very high reliability as they range from BASE to ACID properties
2	Data model	Data model is very specific and well organized. Columns and rows are described by well-defined schema	Data model does not use the table as storage structure and is schema less. Data model is very efficient in handling unstructured data as well
3	Scalability	Scalability is the greatest challenge in relational databases due to the dependency on vertical scalability	NoSQL databases depend on horizontal scalability
4	Cloud	Not well suited for cloud environment as they do not support full content data search. Relational databases are also very hard to scale beyond a limit	Well suited for cloud databases. All characteristics of NoSQL databases (Padhy and Patra, 2011) are highly desirable for cloud databases

Continued

S. No.	Point of Difference	Relational Databases	NoSQL Databases
5	Handling Big Data	Big Data handling is a challenging issue for relational databases	NoSQL databases are designed to handle Big Data
6	Data warehouse	When size of stored data increases, problems related to performance degradation raises	NoSQL databases are not designed to serve data warehouse. NoSQL databases are faster than data warehouse
7	Complexity	Complexity arises due to nonfixture of data into tables	NoSQL databases have the capabilities to store unstructured data
8	Crash recovery	They guarantee crash recovery through recovery manager	NoSQL databases use replication method as backup to recover from crash
9	Authentication	Relational databases come with authentication mechanism	NoSQL database does not come with authentication mechanism, but they can use some external method for this
10	Data integrity	Relational databases ensure data integrity	Data integrity is not always achieved in NoSQL databases
11	Confidentiality	Data confidentiality is often achieved in relational databases	Generally data confidentiality is not achieved in NoSQL databases
12	Auditing	Relational databases provide mechanisms to audit database	Most of the NoSQL databases do not provide mechanism for auditing the database

13.4 Security Threats with NoSQL Databases

NoSQL databases were designed to handle large datasets to meet the requirement of data analytics, and less emphasis was given on security during the design phase. Most of the NoSQL databases do not provide embedded security features in the database itself, and developers need to entail security in the middleware. NoSQL databases provide a very thin layer of security, and to make NoSQL databases secure, the user has to overcome the below-mentioned security threats. These security threats are shown in [Figure 13.1](#).

13.4.1 Distributed Environment

In a distributed environment, the database is distributed over a computer network that allows applications to access data from local and remote databases (Patil and Mukhtar, 2011). A centrally controlled distributed database periodically synchronizes all the data and ensures that updates and deletes are performed on the data and are automatically reflected in the data stored elsewhere.

In a distributed environment, there are several distributed nodes on which NoSQL databases run. These parallel nodes increase the attack surface which makes the system complex to secure. The possibility of unauthorized access to the database increases due to multiple entry points. These entry points may be from a remote location or from a client home location as shown in [Figure 13.2](#).

NoSQL databases segment data horizontally and share them across multiple servers. Data from multiple nodes travel to and from in NoSQL database environment which is distributed across multiple servers.



FIGURE 13.1
Security threats with NoSQL databases.

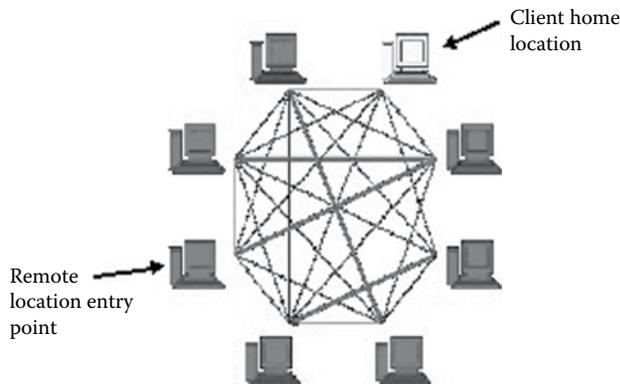


FIGURE 13.2
Security threats with distributed environments.

The maintenance of replicated wreck of data is computationally expensive and more prone to error and also increases the risk of theft. Distributed environment is generally more prone to security risks because of the lack of central security management system. Cassandra and Dynamo databases are vulnerable to security risk in distributed environment because of Gossip membership protocols.

13.4.2 Authentication

An NoSQL database enforces authentication mechanism at local node level but fails to enforce authentication across all commodity servers (Lior, 2011). Due to this, NoSQL

databases are exposed to brute force, injection, and relay attacks, which lead to information leakage. The reasons for these attacks are weak authentication mechanism.

In NoSQL database environment, Kerberos can be used to authenticate the clients and the data nodes, but malicious clients may gain unauthorized access by duplicating the Kerberos ticket. Strict algorithms need to be designed to enforce strict authentication norms.

13.4.3 Safeguarding Integrity

Safeguarding integrity is a security requirement that specifies the extent to which the any database shall ensure that data are protected from unauthorized modification through insertion, deletion, or update. NoSQL databases lack confidentiality and integrity, and to safeguard integrity in these databases is a major concern. The heterogeneous nature of NoSQL databases makes it difficult to protect the integrity of data. As NoSQL databases do not have a schema, permissions on a table, column, or row cannot be segregated. This can lead to multiple copies of the same data, which makes it hard to keep data consistent, particularly as changes to multiple tables cannot be wrapped in a transaction where a logical unit of insert, update, or delete operations is executed as a whole (Michael, 2016). So, maintaining transactional integrity is also very difficult in NoSQL databases. Because of the intricacy in enforcing integrity constraints to NoSQL databases, these databases can never be used for financial transactions. The absence of central control and the schema-less nature of these databases make it difficult to enforce integrity constraints.

13.4.4 Fine-Grained Authorization and Access Control

Authenticating users is the first step in protecting the data. Once the identity of the user is verified, access is granted to the user for part of or the entire database. Authorization plays a very important role in any database. Relational databases store related data and allow authorization at table level. NoSQL databases have no schema and store heterogeneous data together. Therefore, it is difficult to implement authorization on a table as a whole. Fine-grained authorization enables object-level security. Object-level security is further classified into row-and fields-level security. If the data are stored in tables, then row- or cell-level security is applied. If the data or metadata are entered in forms, field-level security is applied and so on. Fine-grained access control is not allowed due to schema-less nature of NoSQL databases. Most of them allow column family-level authorization. The table below demonstrates the type of NoSQL databases with their granularity level.

In NoSQL database, data are grouped according to their security level. The data may be classified as confidential, secret, or even may not be classified at all. So, we need to implement access control row wise or column wise. The same concept is demonstrated in [Figure 13.3](#).

RBAC is difficult to implement as NoSQL database has schema-less structure. Different types of data are stored in one big database in these types of databases. As heterogeneous data are stored together in one database in comparison to relational models, this becomes a challenge.

13.4.5 Protection of Data at Rest and in Motion

Data at rest means data that have been flushed out from the memory and written to the disk. Data in motion means data that are in communication or are being exchanged during a communication. Data in motion are categorized into two categories:

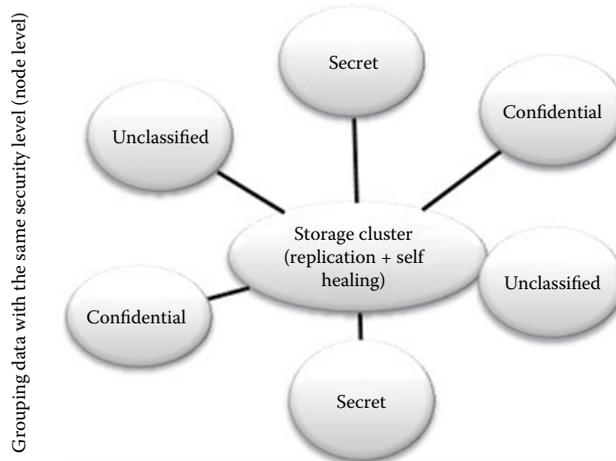


FIGURE 13.3
Fine-grained (row- or column-level) access control.

1. Client-node communication
2. Internode communication

Most of the NoSQL databases do not employ any technique to protect the data at rest. Only a few provide encryption mechanisms to protect data. To safeguard the data in storage, encryption techniques are used and are referred to as de facto standards of encrypted data. Encryption makes the data unintelligible (Lior, 2011) and hence of no use to malicious intruder. Most of the industry solutions lack horizontal scaling while offering encryption services.

The popular NoSQL databases offer following encryption services for protection of data:

1. *Data at rest:*
 - a. Cassandra uses transparent data encryption (TDE) technique to protect data at rest. This feature helps to protect data at rest. This feature helps to protect sensitive data. In Cassandra databases, encryption certificates are stored locally, so a secured file system is required to implement TDE. Also, the commit log of Cassandra database is not encrypted, which also leads to breach of security.
 - b. MongoDB does not provide any method to encrypt the data file. Data files can be encrypted at application layer before writing the data to the database that require strong system security.
2. *Data in motion:*
 - a. Client-node communication: In Cassandra, client-node communication is not encrypted. Encryption is done by generating valid server certificates at SSL layer. MongoDB does not support SSL client-node communication. To encrypt the data using SSL client-node communication, MongoDB needs to recompile by configuring SSL communication.
 - b. Internode communication: Cassandra does not support encrypted internode communication. Internode SSL communication can be configured by editing the corresponding settings under server encryption options in the `Cassandra.yaml` file.

MongoDB does not support internode communication at all.

13.4.6 Privacy of User Data

Privacy of user data is the main challenge for Web 2.0 and NoSQL databases. NoSQL stores a large amount of sensitive information. Maintaining privacy of this information is the key concern for any database administrator.

Clients access NoSQL databases through various nodes and resource managers. Even if a single location is compromised, then the malicious data will propagate to the entire system as there is no central security management. Distributed nature of database also leads to compromised security.

13.5 NoSQL Security Reference Architecture

13.5.1 NoSQL Cluster Security

NoSQL database considers a different method for solving the problems related to Big Data. Virtually, NoSQL does not provide any security within the NOSQL cluster as security of database and data are also dependent on the network and the applications that form a protective shell around the data. This security model is easy to implement as it will not result in performance or functional degradation. Disadvantage of using type of security model is misuse of data by credentialized user or exposure of system to the malicious user if application or firewall fails. Some of the built-in security tools available in NoSQL cluster are SSL/TLS for secure communication, Kerberos for node authentication, data at rest security using transparent encryption, etc. However, these tools of cluster security are difficult to implement and expensive. The intrinsic architecture and design considerations have been carefully taken into account in most of the NoSQL databases. Figures 13.4 and 13.5 represent the core architectural difference between RDBMS databases and NoSQL databases.

A typical NoSQL database makes compromises on some ACID properties. In general, complete security architecture must cover the following points:

- 1. User access management
- 2. Logging operations

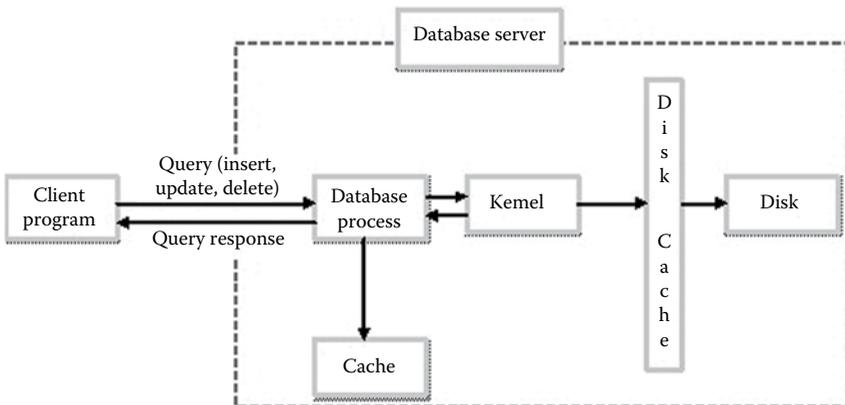


FIGURE 13.4
RDBMS architecture—ACID complaint.

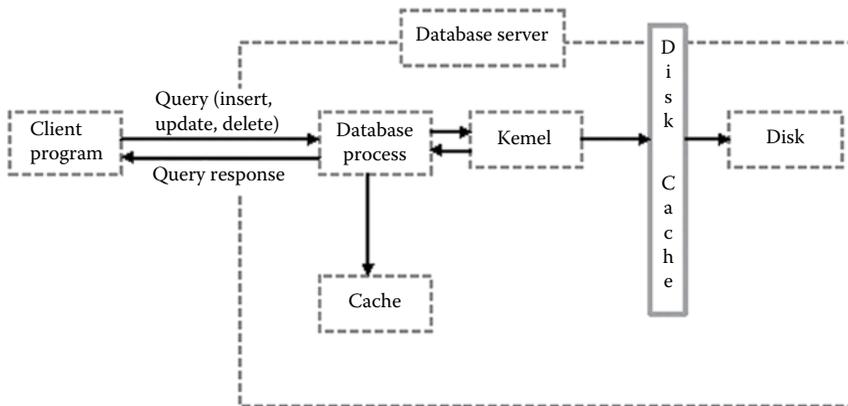


FIGURE 13.5
NoSQL—fire and forget.

3. Data protection
4. Environmental and process control

13.5.1.1 User Access Management: Authentication

Authentication is the process of identifying an individual based on a username and password. In this context, various entities are as follows:

1. Users needing access to database
2. Administration
3. Software systems
4. Physical and logical nodes

Some of the best practices used for user access management are as follows:

1. Create login credentials to avoid creation of a single admin login, which is shared by all users, and a separate security credential is given to each user.
2. Centralized user access management, providing ability to databases to manage authentication within the database itself. It may be done by integrating organizational identity management system.
3. Enforcing password policies, adhering at least minimum password complexity requirements stated by the organization.

13.5.1.2 Authorization

Authorization is defined as the process of giving individual access to the system objects based on the user identity. Authentication purely ensures that the individual is the same person who is claiming to be but has no clue about access right of the individual. Best practices of the authorization include the following:

1. Granting minimum access to entities
2. Grouping common access privileges into roles

3. Controlling actions of individual entity
4. Controlling access of sensitive data

13.5.1.3 Auditing

The process of auditing helps to detect the attempts of accessing unauthorized data. By creating audit trails, logs can be used for compliance. Best practices of auditing include the following:

1. Tracking the changes in database configuration
2. Tracking changes in data

13.5.1.4 Encryption

Encryption is defined as the translation of data into a secret code. It is said to be the most effective way to achieve data security. Some of the best practices used for encryption are as follows:

1. Enforcing connections to databases
2. Encrypting data at rest
3. Enforcing strong encryption
4. Signing and rotating encryption keys

13.5.1.5 Environmental and Process Control

Protection of the underlying infrastructure is also very important. To ensure this, the following best practices are used:

1. Installation of firewalls
2. Network configurations
3. Defining file permissions

13.5.2 Data-Centric Approach to Security

As discussed earlier, NoSQL security is provided either by network or applications that surround the data or by the third-party tools available in NoSQL cluster. Another approach to ensure security in NoSQL databases is data-centric security where security controls are part of data not the database. It means that protection of data takes place before data are moved into the database. Three basic tools that support data-centric security are discussed in [Figures 13.6](#) through [13.8](#) as follows:

1. Tokenization
2. Masking
3. Data element encryption

To ensure tokenization security, sensitive data are substituted with data tokens to ensure security. A token has no basic value; it only carries a reference to the original value to the database. Credit card processing systems use tokenization technique to substitute credit card numbers.



FIGURE 13.6
Tokenization.



FIGURE 13.7
Masking.

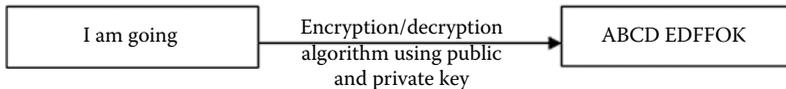


FIGURE 13.8
Data element encryption.

Masking technique replaces original value of data with a random value to protect the data but preserve the original value of the data set for analysis, for example, name of a person can be changed with any other name and date of birth can be modified.

Data element encryption encrypts the data, and only authorized users can decrypt the data using keys.

13.6 Security Weaknesses of Cassandra and MongoDB

Cassandra brings together the distributed system technologies from Amazon DynamoDB (based on the principles of Dynamo) and the data model from Google's Big Table (Deka, 2014). Cassandra is eventually consistent and based on a peer-to-peer (P2P) model without a single point of failure. Like Big Table, Cassandra provides a column family-based data model richer than typical key-value systems (Abramov). Cassandra supports transaction logging and automatic replication.

Security weaknesses of Cassandra are as follows:

1. Cassandra databases are vulnerable to security risk in distributed environment because of Gossip membership protocols.
2. Cassandra uses TDE technique to protect data at rest. This feature helps to protect data at rest. This feature helps to protect sensitive data. In Cassandra databases, encryption certificates are stored locally, so a secured file system is required to implement TDE. Also, the commit log of Cassandra database is not encrypted, which also leads to breach of security.
3. Cassandra does not support encrypted internode communication. Internode SSL communication can be configured by editing the corresponding settings under server encryption options in the `Cassandra.yaml` file.
4. Data in Cassandra are kept unencrypted. Cassandra does not provide a mechanism to automatically encrypt the data in storage making the data vulnerable to attack (Figure 13.9).

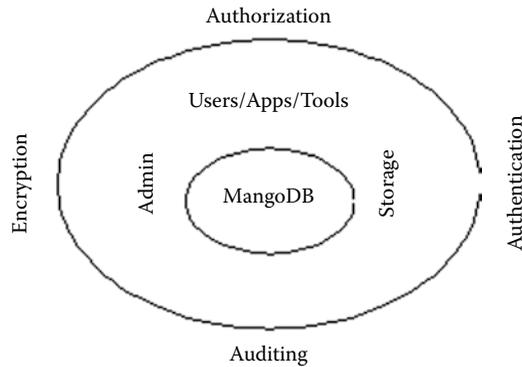


FIGURE 13.9
MongoDB security architecture.

5. Cassandra uses the Apache Thrift framework for client communications which uses SSL transport. The client interface supports a login () operation (Deka, 2014), but both username and password are sent across the network as clear text.
6. In Cassandra, nodes on a cluster can communicate freely, and no encryption or authentication is used. All communication between the database and its clients is unencrypted. An attacker capable of monitoring the database traffic will be able to see all of the data as the clients see it.
7. Cassandra uses a new interface called Cassandra Query Language (CQL) (Deka, 2014). CQL syntax is similar to SQL. CQL is compatible with the JDBC API. Being a parsed language, CQL is vulnerable to injection attacks, like SQL.
8. Cassandra uses a thread-per-client model in its network code. Setting up a connection requires the Cassandra server to start a new thread on each connection. An attacker can prevent the Cassandra server from accepting new client connections by causing the Cassandra server to allocate all its resources to fake connection attempts.
9. Cassandra provides an IAuthenticate interface. The default implementation is one that turns off the requirement to authenticate to the database, which makes Cassandra vulnerable to attack.
10. Cassandra provides an IAuthority interface which is used in the Cassandra's codebase when a keyspace is being modified, and on each column family access (read or write). The IAuthority interface provides a single method returning a set of permissions for a provided authenticated user and hierarchical list of resource names.
11. Cassandra does a custom implementation of IAuthority. IAuthority can be written to provide full auditing of all operations that require authorization, and a custom implementation of IAuthenticate can be written to provide a full audit trail for all login success/failure occurrences. Cassandra aims to bring the risk-based audit (RBA) approach to a next level. RBA approach focuses on the risks and the underlying causes of disruptions in transaction results as well as records. Therefore, RBA approach shifts the attention from transaction quality to overall process quality.

MongoDB, classified as a NoSQL database, is a cross-platform, document-oriented database. It is an open-source database and has been written in C++ language. It provides high availability, high performance, and easy scalability. MongoDB database uses the concept of document and collection (MongoDB, 2016). A collection is said to be a group of MongoDB documents like an RDBMS table.

1. MongoDB does not support SSL client-node communication. To encrypt the data using SSL client-node communication, MongoDB needs to recompile by configuring SSL communication.
2. MongoDB does not support internode communication at all.
3. MongoDB does not provide any method to encrypt the data file. Data files can be encrypted at application layer before writing the data to the database, which requires strong system security.
4. MongoDB uses Java script as internal scripting language. Intensive use of Java script has made MongoDB susceptible to injection attack.
5. MongoDB does not support authentication in shared mode and therefore no support for authorization as well. Authentication can be implemented in unshared mode.
6. MongoDB suffers denial of service attacks due to the sudden peak in the CPU and memory loads of host systems.

Details of MongoDB security features and measures have also been discussed in Section 13.8.

13.7 Securing NoSQL Applications

Most of the NoSQL databases are open source and founded upon the Hadoop framework. Hadoop framework is not a single technology but is a combination of various cooperating applications. Each of these applications has different security requirements and is modified as per the need of the data. Therefore, to achieve database security, various parameters, as listed below, need to be implemented in combination with each other:

1. Using firewalls
2. Auditing and logging
3. Authentication
4. Input validation
5. Access control
6. Segregation of duties
7. Encryption

It is necessary to implement the security mechanism within the NoSQL database environment rather than on a control point from where data and queries enter the database

environment. Although we have discussed all the listed parameters earlier as well, a brief description is as follows:

1. **Firewalls:** Firewalls are used to protect the system from malicious and intentional attacks. Firewalls can be embedded close to the data store to strengthen the user authentication process. Once the authentication process is strengthened, other security factors will automatically be reinforced.
2. **Audit and logging:** Auditing of the databases needs to be done very frequently to identify mishaps as early as possible. Third-party tools like Scribe and Logstash can be integrated into the database to log the records of transactions. Auditing using Scribe helps in identifying theft of data, and logging the transaction records will help in saving data misuse.
3. **Authentication:** Authentication can be implemented within the database or within the framework. Authentication keeps a strong check over the malicious user. Strong password mechanisms like user-defined passwords can also be a measure to take care of.
4. **Input validation:** Input validation is required to filter Java scripts, which will help in the elimination of JavaScript injection attacks and string concatenation.
5. **Access control:** Access control helps in imposing restriction on data access. Access to data can be granted on the basis of security policy of the organization and the permission levels.
6. **Segregation of duties:** Access to data can be granted on the basis of role and responsibility of the employee. Segregation of duties will help in restricting data theft and malpractices.
7. **Encryption:** It is required to transform the data into a format which can only be decrypted by users having authorized key. It helps in protecting sensitive data.

13.8 Understanding Security Weakness in a Typical NoSQL Database: MongoDB

MongoDB, classified as an NoSQL database, is a cross-platform, document-oriented database. It is an open-source database and has been written in C++ language. It provides high availability, high performance, and easy scalability. MongoDB database uses the concept of document (MongoDB, 2016) and collection. A collection is said to be a group of MongoDB documents like an RDBMS table. [Table 13.1](#) shows various NoSQL databases and their granularity levels. [Table 13.2](#) shows the terminology of MongoDB and RDBMS.

TABLE 13.1

NoSQL Databases and Their Granularity Levels

NoSQL DBMS	Granularity	Explanation
BigTable	Column family	Using ACL
Cassandra	Column family	Using IAuthorizer API
HBase	Column family/cell	Group-based authorization
Accumulo	Cell	Using visibility field

TABLE 13.2

Terminology of MongoDB and RDBMS

MongoDB	RDBMS
Database	Database
Collection	Table
Document	Tuple/row
Field	Column
Embedded documents	Table join
Primary key as the default key-id given by database itself	Primary key
Data server—MongoD	Data server—MySQL/Oracle
Client—Mongo	Client—MySQL/SQLPlus

Following example represents the document structure of a blogging site:

```
{
  _id: ObjectID (7fd87da2089b)
  Title: "Overview of MongoDB"
  Url: "nosql-databases.blogspot.com"
  Tags: ['mongodb', 'database', 'NOSQL']
  Likes: 225
  Comments: [
    {
      User: 'user25',
      Message: 'useful database',
      Datecreated: new date (2016, 1, 18, 2, 10),
      Like: 1
    },
    {
      User: 'user35',
      Message: 'I use it',
      Datecreated: new date(2016,1,19,3,10),
      Like: 0
    }
  ]
}
_id gives the uniqueness to every document as it is a 12 digit hexadecimal number.
```

13.8.1 Advantages of MongoDB over RDBMS

1. A relational database has a fixed schema design which represents the number of tables and relationship between these tables, whereas in MongoDB, there is no concept of relationship, and hence there is no complex joins in MongoDB database.
2. MongoDB is schema less as it is a document database in which one collection holds different types of documents. The content, number of fields, and size of document may be different in different documents.
3. MongoDB supports dynamic query on document using a document-based query language.

4. MongoDB database is easy to scale.
5. Mapping or conversion of application objects to database objects is not required in MongoDB.
6. MongoDB uses internal memory for storing the worrying set, and hence it enables the faster access of data.

13.8.2 MongoDB Security Architecture

MongoDB has advanced security features which are the basic requirement of the user these days. Some of the foundational security requirements of MongoDB are as follows:

1. Restricted access to data can be enforced by providing pre-defined privileges to user and by creating various security levels
2. Protection of sensitive data from malicious user from accidental damage
3. Maintaining record of activities of users, applications, and administrative staff while accessing and processing data

Based on the following requirements, holistic security architecture of MongoDB databases has been developed which is given below. This architecture helps in implementation of a secure, compliant data management environment.

The security architecture of MongoDB is covering the following points:

1. *User access management: Authentication*

Authentication helps in identification of the entities and accessing the database. MongoDB supports authentication in following ways:

- a. Supporting in-database and centralized user access management
- b. Enforcing password policies

2. *User rights management: Authorization*

MongoDB supports the following practices for authorization:

- a. Granting minimal access to entities
- b. Controlling the actions an entity can perform
- c. Common access privileges to be grouped into roles
- d. Controlling the access to sensitive data

3. *Auditing*

Auditing helps in detection of attempts made to access unauthorized data. Best practices of auditing supported by MongoDB are as follows:

- a. Track changes to database configuration
- b. Track changes to data

4. *Encryption*

MongoDB supports encryption with the following practices:

- a. Encrypt connection to the database
- b. Enforce strong encryption
- c. Encrypt data at rest
- d. Sign and rotate encryption keys

Apart from above-mentioned security features of MongoDB databases, additional security protection measures that can be implemented in deployment environment of MongoDB database to make it secure are as follows:

1. Installation of firewall
2. Network configuration
3. Defining file system permissions
4. Physical access controls to the IT environment

13.8.3 Security Measures in MongoDB

The following points provide a list of security measures that one must implement to protect MongoDB installation:

1. Enable access control
2. Enforce authentication
3. Configure RBAC
4. Encrypt communication
5. Limit network exposure
6. Audit system activity
7. Encrypt and protect data
8. Run MongoDB with a dedicated user
9. Run MongoDB with secure configuration option
10. Consider security standards compliance

References

- Ahmed, J. and Gulmeher, R. 2014. NoSQL databases: New trend of databases, emerging reasons, classification and security issues. *International Journal of Engineering Sciences & Research Technology*, 4(6), 176–184.
- Deka, G. C. (ed.). 2014. Cloud database security issues and challenges. In *Handbook of Research on Securing Cloud-Based Databases with Biometric Applications*. IGI Global, USA.
- Ebrahim, S. and Mohammad, A. N. 2015. Survey on security issues in Big Data and NoSQL. *ACSII Advances in Computer Science: An International Journal*, 4(4), 68–72.
- Factor, P. 2013. NoSQL: Are You Ready to Compromise with Security. Available at: <http://www.sqlservercentral.com/articles/Editorial/NoSQL+NoAuthorisation/98005/>
- Lior, O. 2011. Security issues in NoSQL databases. In *International Joint Conference of IEEE, Changsha, Hunan Province, P. R. China. TrustCom-11/IEEE ICSS-11/FCST-11*.
- Michael, C. 2016. NoSQL Security: Do NoSQL Database Security Features Stack Up to RDBMS? Available: <http://searchsecurity.techtarget.com/tip/NoSQL-security-Do-NoSQL-database-security-features-stack-up-to-RDBMS>. Retrieved on April 10, 2016.
- Mohamed, A. and Obay, G. 2014. Relational vs. NoSQL databases: A survey. *International Journal of Computer and Information Technology*, 3(3), 598–601.
- MongoDB. 2016. Mongoddb. [Online] Available: <http://www.mongodb.org/>. Retrieved on April 15, 2016.

- Nance, C. and Losser, T. 2013. NOSQL VS RDBMS—Why there is room for both. In *Proceedings of the Southern Association for Information Systems Conference*, Savannah, Georgia, March 8–9, pp. 111–116.
- Padhy, R. P. and Patra, M. R. 2011. RDBMS to NoSQL: Reviewing some next-generation non-relational databases. *International Journal of Advanced Engineering Science and Technologies*, 11(2), 45–52.
- Patil, H. S. and Mukhtar, Y. S. 2011. Distributed database: An relevance to business organization. *Journal of Information and Operations Management*, 2(1), 21–24.
- Tabrez, Q. 2013. Security issues in distributed database system model. *COMPUSOFT, An International Journal of Advanced Computer Technology*, 2(12), 396–399.

14

Security Issues and Privacy Challenges of NoSQL Databases

Mohammad Samadi Gharajeh

CONTENTS

14.1 Introduction of NoSQL Databases	271
14.1.1 Introduction to Database Security	272
14.1.2 Database Auditing and Monitoring	273
14.1.3 Key Database Security Functions	274
14.1.4 Architectural Security for NoSQL Systems	275
14.1.5 Third-Party Tools versus Database Vendors.....	277
14.2 Security Issues of NoSQL Databases	277
14.2.1 Security in the Popular NoSQL Databases	279
14.2.2 Model-Based Security-Aware Elasticity	283
14.2.3 Security Problems of the NoSQL Environments.....	283
14.3 Privacy Challenges of NoSQL Databases.....	285
14.3.1 The Privacy of Data Owners and Query Users	286
14.3.2 Solutions for Existing Privacy Problems	287
14.4 Comparison Results.....	288
References.....	289

14.1 Introduction of NoSQL Databases

Not only SQL, called NoSQL, database is one of the trending terms in modern data storage systems. Compared to relational database, it relies on different storage strategies including key-value store, document store, and graph. The main goal of these databases is to distribute a high volume of data across a huge number of cloud servers. This goal cannot be achieved by traditional databases because the scalability requirements are not met by them. When NoSQL databases were emerged for the first time, they were lacking security requirements, similar to any new technology. Hence, the lack of appropriate encryption, authentication, role management, and grained authorization caused some problems to these databases. But now, the popular NoSQL databases contain built-in protection mechanisms to prevent security attacks (Han et al. 2011; Grolinger et al. 2013; Rocha et al. 2015).

In this book chapter, I will discuss on the security issues of NoSQL databases in aspects of security in the popular NoSQL databases, model-based security-aware elasticity, and security problems of the NoSQL system environments. Furthermore, the existing privacy problems for NoSQL data storages are explained according to data storage, data manipulation, and data query.

14.1.1 Introduction to Database Security

There are various security issues about big data. Some of the security and privacy challenges in big data are presented in Mora (2012). Secure computation is one of the major challenges in distributed systems that discusses on security issues in map-reduce functions. Secure data transactions describe new mechanisms to prevent unauthorized access to data storages and to maintain availability of the services to anyone. Granular access process is another challenge in big data that prevents access to data by the users who are not permitted to have access. In this case, most of the traditional access models have major constraints for dealing with big data.

Efficiency of the security in NoSQL databases is compared to that in the relational databases (Maier 1983).

As represented in [Table 14.1](#), the security features of relational and NoSQL databases are compared to each other based on authentication, data integrity, confidentiality, auditing, and client communication (Mohamed et al. 2014). Authentication confirms the identity of the users and systems over communicating areas. It forms the basis for cloud security whether in the cloud or on the local network. Many NoSQL databases do not come with any authentication strategy, but they use some external tools to perform the authentication process. Data integrity determines the accuracy and consistent features of stored data in data modification between two updates of the data record. To ensure cloud data storage, cloud clients store their data in the cloud so that server no longer maintains the data locally in cloud. Relational databases use Atomicity, Consistency, Isolation, and Durability (ACID) to guarantee data integrity with the aid of reliable transactions. Data confidentiality in cloud storages protects access to the data. It assures that certain information would not be disclosed without permission from the subject or the sponsor. The confidentiality implies that customers' data and computation tasks are kept confidential from both cloud providers and other customers. Data confidentiality is not always implied in NoSQL databases because these databases use clear data without having any encryption technique. The goal of auditing in cloud-based storages is to offer cloud service providers in a way that their performance and security are available to potential customers. It offers a standard way to share automated statistics about performance and security. Auditing is not provided by most of the NoSQL databases and is only provided by some databases (e.g., CouchDB) to store usernames and passwords in log files. Client communication in cloud systems performs data communication between clients and cloud servers. The protocols for client communication may be binary (e.g., CORBA) or XML (e.g., SOAP). Most of the NoSQL databases do not provide secure client communication.

TABLE 14.1

Security Categories in Relational and NoSQL Databases

Category	Relational Databases	NoSQL Databases
Authentication	Authenticated by all relational databases	Not authenticated by many NoSQL databases
Data integrity	Guaranteed by ACID properties	Not always achieved
Confidentiality	Obtained often	Not always obtained
Auditing	Available by most of the relational databases	Available by some of the relational databases
Client communication	Possible by secure client communications	Not possible by most of the NoSQL databases

14.1.2 Database Auditing and Monitoring

The auditing and monitoring features of the NoSQL databases are used to specify “who, what, where, and when” of the cloud users access databases at different places. They apply third-party open-source tools to integrate these features into most of the big data environments (e.g., Scribe and LogStash). The integration process is performed via an interaction with other systems (e.g., log management). Audits of the NoSQL databases are used to discover mishaps, quickly. The auditing process is available on Enterprise Cassandra database to get the maximum audit information on every node. Furthermore, the security logging and monitoring are used in MongoDB database to show the running state of the database instance. The information about system and connected clients is available by an HTTP Console for each MongoDB instance (Fidelis Cybersecurity 2014).

Nowadays, the Worldwide LHC Computing Grid (WLCG) plays a major role in more than 150 computing centers to execute more than 2 million job accounting. The WLCG monitoring of data using NoSQL is represented in Andreeva et al. (2014). It describes the integration of NoSQL data processing into the Experiment Dashboard framework and some cluster specifications. In the experiment results, Oracle is compared to Elasticsearch to execute job accounting as represented in Table 14.2. Note that Elasticsearch is an alternative schema suggested by the CERN Agile Infrastructure (AI) Monitoring team. The comparison results indicate that Elasticsearch has a high performance compared to an Oracle cluster. The comparison process is performed on different scan times with or without any filters.

Figure 14.1 shows an NoSQL-based data management infrastructure for bridge monitoring database (Jeong et al. 2015).

As shown in Figure 14.1, it consists of an on-site computer, a local computer, a main server, and a web interface for application tools and users. The sensors, which are placed on the bridge, transmit the phenomena information to the on-site computer. In the second stage, the on-site computer stores the data on the database, temporarily, and then, if appropriate, it performs real-time analysis. Afterward, the sensing data are parsed and sent to the main server. All sensing data and bridge information (e.g., bridge geometry) are persistently stored on the main server. A local computer enables the application tools and users

TABLE 14.2

Comparison Results of the Job Accounting in Oracle and Elasticsearch

Period	Scan Type		Oracle	Elasticsearch
	Filter	Average Rows	Scan Time (s)	Scan Time (s)
1 day	0	116	0.31	0.017
1 week	0	807	0.2	0.118
1 month	0	3600	0.956	0.138
2 months	0	7000	2.27	0.16
1 day	1	13	0.013	0.016
1 week	1	98	0.018	0.021
1 month	1	431	0.101	0.056
2 months	1	864	0.16	0.062
1 day	2	5	0.01	0.003
1 week	2	28	0.013	0.004
1 month	2	123	0.055	0.031
2 months	2	259	0.101	0.097

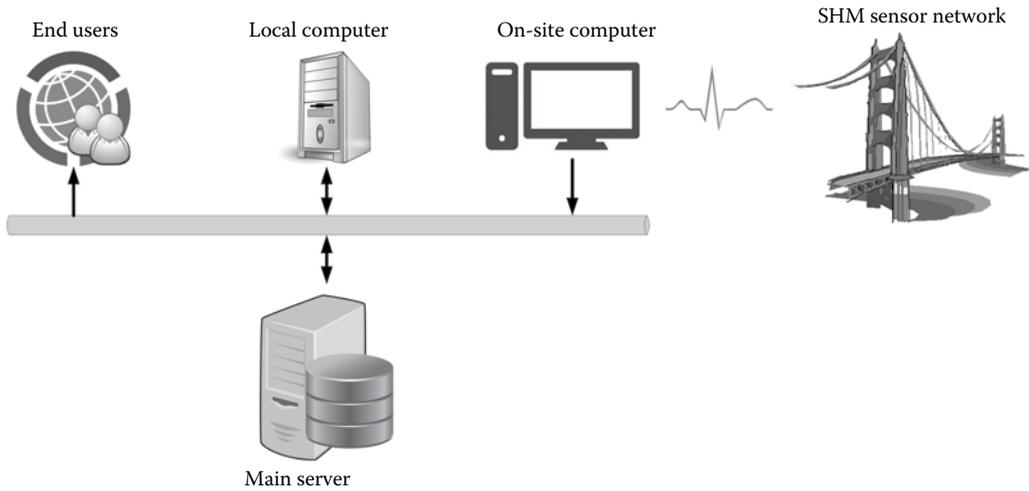


FIGURE 14.1
An overall view of the NoSQL-based data management infrastructure for bridge monitoring.

to get the data from the main server. The end users (e.g., bridge managers) can obtain the analysis results of the bridge status via a web-based interface.

14.1.3 Key Database Security Functions

The key database security functions are represented in [Figure 14.2](#). They are grouped into four categories including vulnerability assessment and scanning, database auditing and monitoring, real-time protection and database firewalls, and database encryption.

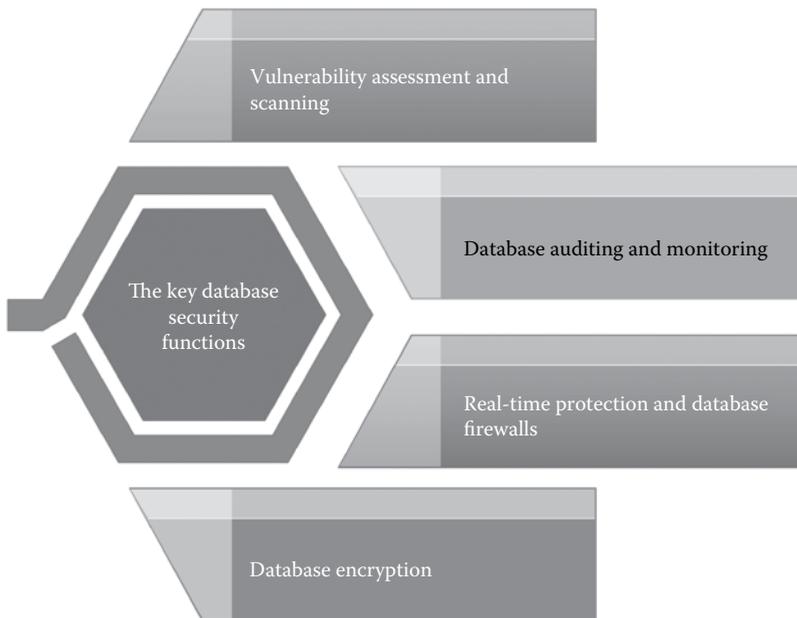


FIGURE 14.2
The key database security functions to take the secure databases.

and monitoring, real-time protection and database firewalls, and database encryption (Database security: At rest, but not at risk 2016). These functions play major roles in enhancing the security levels of the database. Any database designer must consider them to carry out the secure databases.

Vulnerability scanners are the most mature category of the database security tools that report on various risks including default passwords, stale accounts, outdated patches, unwarranted user privileges, incorrect configurations, and more. Corporations, growingly, have more interests to track and manage different activities of privileged users. A major problem of the scanner is to feedback an unpredicted number of results. The existing solution for this problem is to start with the easiest management parameters (e.g., blank passwords) and then to move to another issue (e.g., default passwords). The management process of the output from scanner report is a big challenge when some of the vulnerabilities (e.g., patching) cannot be addressed carefully.

Real-time protection and database firewalls are another category of the security functions. Corporations have begun to move into the real-time database protection. These tools look for well-known attacks (e.g., SQL injections) to automatically block or quarantine such attacks and suspicious behaviors (e.g., the user accessing a large volume of the records). Compared to automatically blocking or quarantine, corporations intend to use an alert performed by a manual investigation.

Database encryption in the database security functions takes a long term and also is very mature through a security research and advisory firm. The database vendors, usually, provide the encryption mechanisms within the database itself. They utilize some of the third-party tools and intercept files to encrypt or decrypt the databases. The encryption tools are slowly used for databases with the compliance as the main driver for the adoption procedure and the payment card industry. These cases are used in various corporations such as ARC to build some financial tools for the travel industry.

14.1.4 Architectural Security for NoSQL Systems

Architectural security for NoSQL systems discusses on the deployment phase of the big data security. The architectural security model and different deployment features indicate a different procedure to secure big data. Since a big number of built-in security capabilities are not presented from big data assignments, the security challenges in NoSQL databases are more different from those in traditional databases, parallel processing environments, and data warehouses. Big data is recognized by the basically deployment models including highly distributed, redundant, and elastic data repositories which are enabled by the Hadoop File System (HDFS). As shown in [Figure 14.3](#), a distributed file system offers the most essential characteristics and allows the enormous parallel computation. Data nodes communicate with clients and resource management to facilitate the security concerns in the system (Lane 2012; NoSQL Vendor Comparison 2016).

As depicted in [Figure 14.4](#), architectural issues are categorized into multiple items including distributed nodes, “Sharded” data, data access/ownership, internode communication, client interaction, and NoSecurity. Distributed nodes involving various data can be processed anywhere, network resources can be available anytime, and enormous parallel computation can be possible throughout the system. “Sharded” data represent that data within big data systems are fluid with having multiple copies that can be moved to and from different distributed nodes. This is done to guarantee redundant and resilient big data. Data access/ownership enables the role-based access in most of the database security schemes. It limits user access to the authorized subsets of the available big data

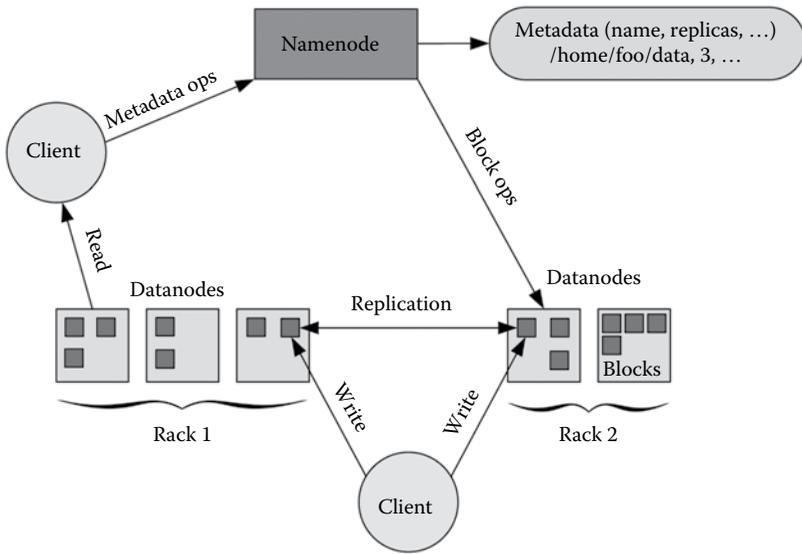


FIGURE 14.3
An overall view of the HDFS architecture.

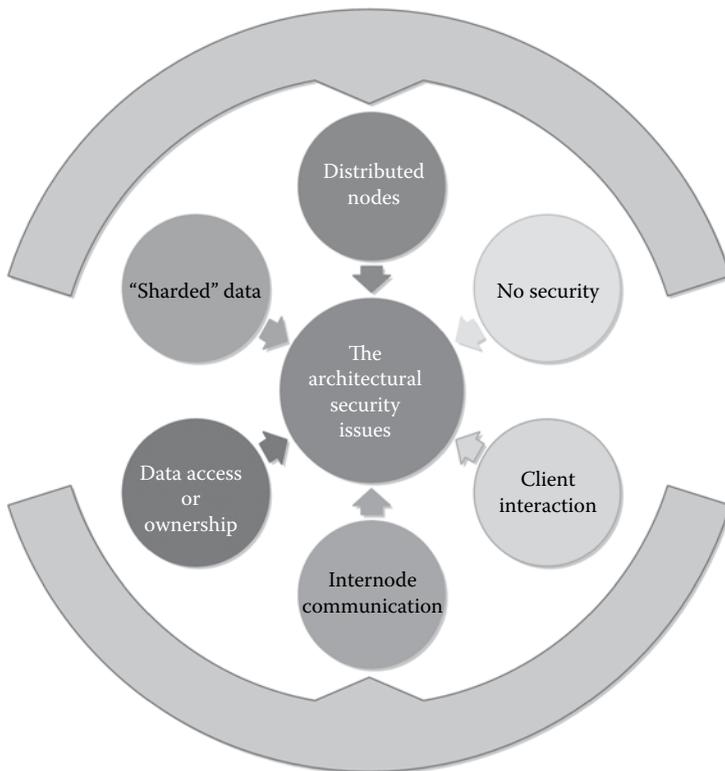


FIGURE 14.4
The architectural security issues in the NoSQL systems.

under different platforms. Internode communication allows Hadoop and the vast majority of distributions (e.g., Cassandra) to cover the client-to-proxy communication across the network. Client interaction indicates the interaction process of the clients with resource managers and nodes. When the gateway services are created for loading data, clients can directly communicate with both resource managers and individual data nodes. Finally, NoSecurity represents that big data stacks, practically, build no security. In this case, there is not any capability to protect data stores, applications, and core Hadoop features.

14.1.5 Third-Party Tools versus Database Vendors

Most of the enterprises apply those of the security features which are natively come with database management system (DBMS). However, they turn to the third-party tools for advanced requirements. The real-time protection, granular compliance reporting, and support for heterogeneous deployments are some of these requirements. The security features of DBMS are isolated per any database vendor and do not offer a holistic view. Therefore, it is requested to go with a centralized tool to provide a complete view of the threat landscape from one console and, also, integrate with other enterprises for management, risk reporting, and analytics. The databases themselves do not involve enough security tools to meet the compliance initiatives for tracking and understanding anything performed by privileged users. The third-party tools can find fewer problems in the databases compared to the database vendors (Database security: At rest, but not at risk 2016).

14.2 Security Issues of NoSQL Databases

This section discusses about security built into the NoSQL database environments. Moreover, it evaluates the major weaknesses of these systems. The main goal of the section is to explain different security problems of the NoSQL database environments and represent the best mechanisms to secure these environments. Various data stored in these systems should be safely secured by security mechanisms. The security evaluation is performed for the top big data vendors and deployment phases in terms of server configuration, storage hardware, database software, analytics applications, and other associated services. Since big data in the hands of the organizations is highly valuable, they should be protected in various security aspects (Kadebu and Mapanga 2014). Security elements for NoSQL databases are shown in [Figure 14.5](#). It consists of security objectives, security mechanisms, security policy, and security models.

Security requirements are needed in NoSQL data storages to satisfy the security goals. They are categorized into several groups including identification requirement, authentication requirement, authorization requirement, immunity requirement, and more. Identification requirement indicates any security requirement to specify the extent to which the database will describe users or entities. Authentication requirement indicates any security requirement to define the extent to which the database system will be capable to make the confirmation of the identity for the entity wishing to access resources. Authorization requirement represents any security requirement to determine the extent to which the database system will verify the user who has the correct permissions and rights to access the requested cloud resource. Immunity requirement indicates any security

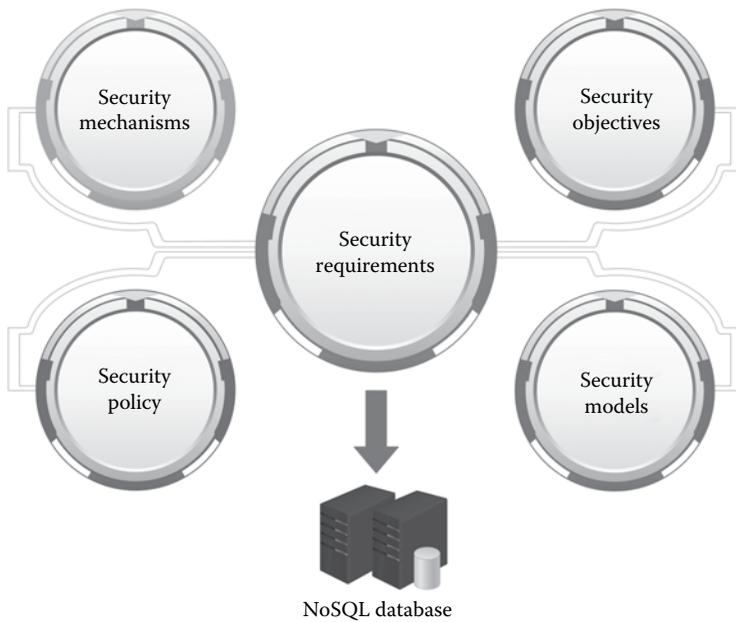


FIGURE 14.5
Security elements for NoSQL database.

requirement to specify the extent to which the database system will offer an internal ability to defend itself from attacks occurred by malicious software.

Security objectives associated to database systems are established by organizations to protect data stores. They express security goals of the system in operational terms. The efficient data storage and retrieval ability are the main objectives of NoSQL databases. Security objectives uncover security problems inherent in the NoSQL database environments. Security mechanisms are considered to obtain the security requirements for the NoSQL databases. They provide those of the security services which are related to the security objectives. Input validation, access control, and encryption are some of the security mechanisms in NoSQL storage systems. Input validation eliminates JavaScript injection attacks and string concatenation. Access control grants access of the authenticated user to special cloud resources based on organizational security policies. Encryption protects the confidentiality or privacy of data as well as it applies encryption algorithms to sensitive data.

Security policy represents how an organization can make plans to protect information of any company with defining the goals, purposes, responsibility, and overall requirements. They identify the rules for maintaining the security in a system and define the security visions for organizations. A security policy, exactly, describes which security level should be set by the security mechanisms to accomplish security procedures. Security models involve some of the statements to indicate some requirements for supporting and implementing a certain security policy. A security model is a symbolic representation of a security policy. It is a framework to give the policy form and solve the security problems in special conditions. A security model indicates main procedures without having any idea about how they would be completed. Biba model, Clark–Wilson model, and Graham–Denning model are some of the security models in NoSQL databases.

14.2.1 Security in the Popular NoSQL Databases

MongoDB, Cassandra, CouchDB, HBase, and HyperTable are the most popular NoSQL databases (Khetrapal and Ganesh 2006; Han et al. 2011; Okman et al. 2011; Boicea et al. 2012; Grolinger et al. 2013; Zahid et al. 2014; HyperTable 2016) Since they are used in the most cloud-based servers, the security on these systems is very necessary. The increasing popularity of these databases raises the concern for the security offered by them. The high security leads to data stored on these systems to be confidential and a high volume of data are hired by cloud users.

MongoDB: This NoSQL database is a schema-free and document-oriented database that manages collections of schema-less JSON-like documents. This process allows big data to be nested in a complex hierarchical structure with having queryable and indexable capabilities. MongoDB has four main features as follows: data model, API, architecture, and sharding. Data model in this database considers holding a set of collections. The API feature involves its own query language, called Mongo Query Language. A query document, containing different fields of the desired document, is created to retrieve certain documents. The architecture of MongoDB is composed of multiple clusters. As shown in Figure 14.6, any cluster consists of one or more shards where each shard maintains a portioned data. The read and write operations are automatically transmitted to the appropriate shard(s). Each shard involves a replica set holding the data for the related shard. Any replica set is composed of one or more servers to hold various copies of the same data. All data in this database are stored as plain text in a way that there is not any encryption strategy to encrypt data files. MongoDB applies the SSL and X.509 certificates to create a secure communication between user and MongoDB cluster through an intracluster authentication. As represented in Okman et al. (2011), MongoDB is very potential for scripting injection attack because it utilizes JavaScript as the internal scripting language. Table 14.3 represents the security features of MongoDB with some recommendations to solve the existing problems (Okman et al. 2011).

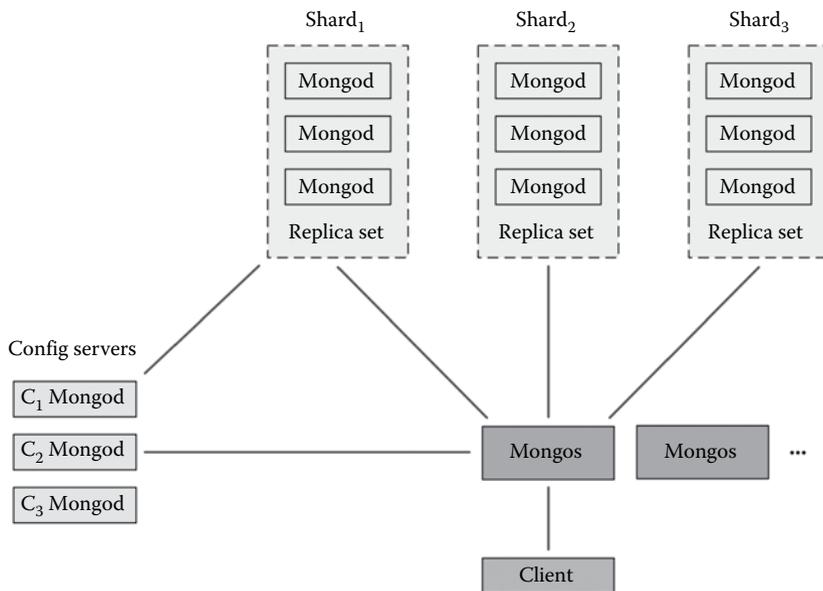


FIGURE 14.6
The internal architecture of the MongoDB database.

TABLE 14.3
The Security Features of the MongoDB Database

Category	Status	Recommendation
Data at rest	Unencrypted	Protect with operating system (OS)-level mechanisms
Authentication for the native connections	Available for unsharded configurations	If possible, enable the authentication strategy
Authorization for the native connections	Read/read-write/admin levels for unsharded configurations	Require the enabled authentication
Auditing	Not available	-
Authentication, authorization, and auditing (AAA) for rest connections	Users and permissions are maintained with external tools	Available if configured on the reversed proxy
Database communications	Available without any encryption mechanism	-
Injection attacks	Possible by JavaScript or string concatenation	Available via reasonable input validation

Cassandra: This NoSQL database is an open-source distributed storage to manage big data. It is a key-value NoSQL database used in the Facebook community site. The properties represented in Han et al. (2011) indicate that Cassandra involves the flexibility of the various schemes and supports the range query with a high scalability. A schematic of this database is illustrated in Figure 14.7. All passwords in the Cassandra database are encrypted with the MD5 hash function. Note that they are set very weak security. If a malicious user bypasses the client authorization, it can obtain the data of the server. The reason is that there is not any authorization mechanism in the internode message exchange. The Cassandra database is potential for the denial of service attacks because it uses one thread per one client and does not support the inline auditing. It applies a query language called Cassandra Query Language (CQL), which operates something similar to SQL. This database can be hacked by the injection attacks and, also, has a major problem in managing the

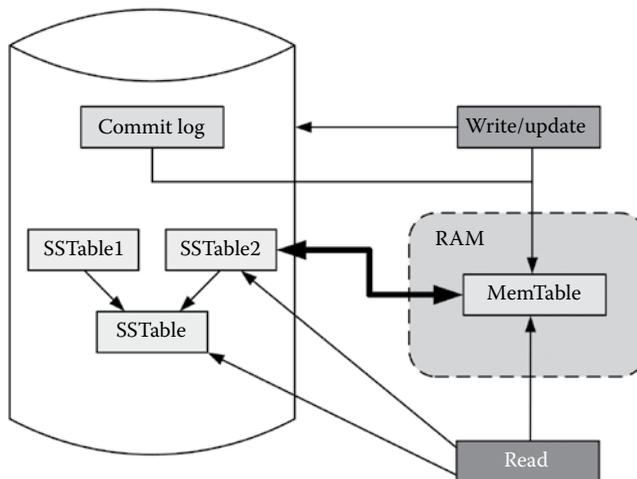


FIGURE 14.7
The architecture of the Cassandra database.

TABLE 14.4
The Security Features of the Cassandra Database

Category	Status	Recommendation
Data at rest	Unencrypted	Protect with OS-level strategies
Authentication	Not available	Implement a custom <i>IAuthentication</i> provider
Authorization	Available at the column family (CF) granularity level	Implement a custom <i>IAuthority</i> provider
Auditing	Not available out of the box (OOTB)	Implement the authentication and authorization solutions
Intercluster network communications	Encrypted	Enable with a private CA
Client communications	Unencrypted	Add packet-filter rules to avoid unknown hosts from the connections
Injection attacks	Possible in CQL	Perform input validation in the application

inactive connections. Table 14.4 represents the security features of Cassandra with some recommendations to solve the mentioned problems (Okman et al. 2011).

CouchDB: This NoSQL database stores the data into documents. Any document has a desired number of attributes so that every attribute, itself, can involve lists or even objects. The documents should be stored and accessed as the JSON objects because CouchDB must support some of the data types including String, Number, Boolean, and Array. Note that, any CouchDB document has a unique identifier. Whereas this database uses the optimistic replication on the server side and the client side, documents have a revision identifier. A schematic of the CouchDB database is shown in Figure 14.8. The CouchDB database does not support data encryption, but it supports some authentication procedures based on

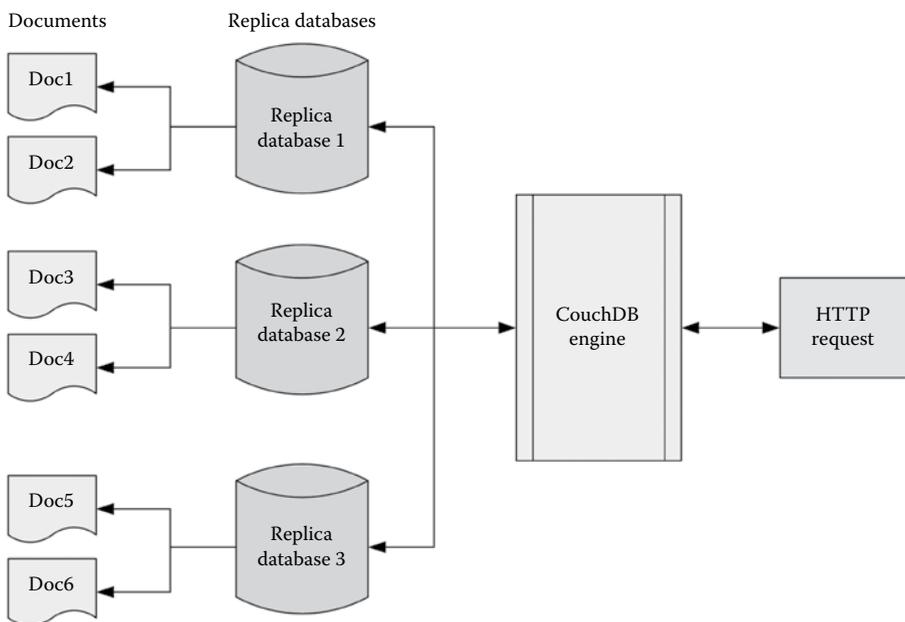


FIGURE 14.8
A simple architecture of the CouchDB database.

both passwords and cookies (Zahid et al. 2014). Any password is encrypted by the PBKDF2 hash algorithm and is transmitted across the network by the SSL protocol. It is worth noting that the CouchDB database is potential for denial of service attacks and script injection.

HBase: This NoSQL database is an open-source toolkit which is written in Java and developed by the Apache Software Foundation. It works with Apache’s HDFS as the basic data storage. HBase is used in the high-performance real-time queries for very huge amounts of the distributed data. The data model of this database is stated as follows: (i) any table consists of rows and columns, (ii) each column belongs to a certain CF, (iii) every row is identified by the unique key, and (iv) a table cell is made by the intersection of a row and a column (Khetrapal and Ganesh 2006). A schematic of the HBase database is shown in Figure 14.9. The Hbase database uses the SSH protocol for internode communications. It supports the user authentication by the Simple Authentication and Security Layer (SASL) with Kerberos. Furthermore, it supports the authorization procedure by access control list (ACL) (Zahid et al. 2014).

HyperTable: This NoSQL database is a high-performance, open-source, and massively scalable database. It represents data as the tables of information, with rows and columns. The row keys are UTF-8 strings as well as it does not support data types, joins, and transactions. Figure 14.10 illustrates the internal architecture of the HyperTable (2016) database). This database does not support the data encryption and authentication processes. It cannot recover the lost data when a range server crashes over the network. Although the HyperTable database uses Hypertable Query Language (HQL), it does not involve

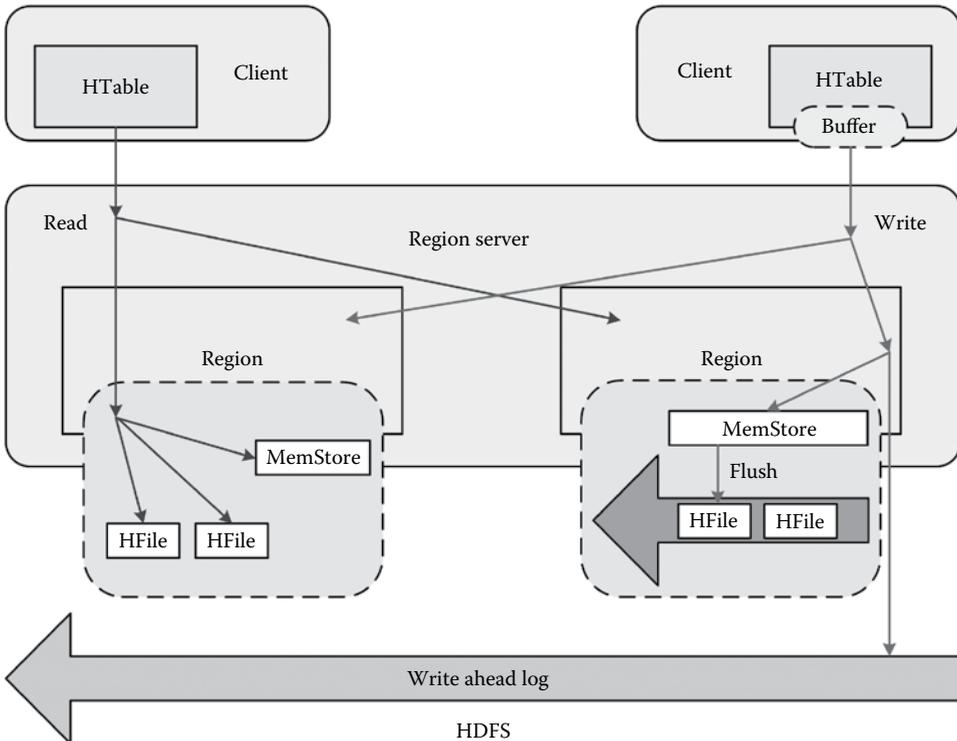


FIGURE 14.9
The internal architecture of the HBase database.

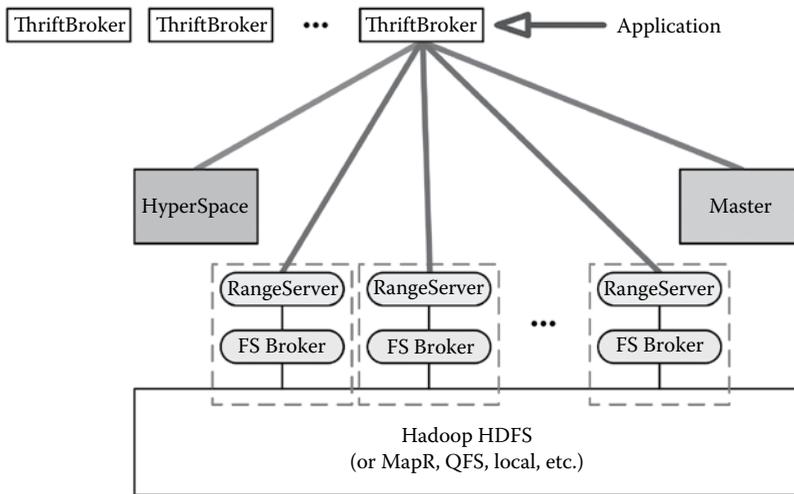


FIGURE 14.10
A schematic of the HyperTable architecture.

any vulnerabilities for the injection process. Note that, there is not any denial of service reported for this NoSQL database (Khetrapal and Ganesh 2006).

14.2.2 Model-Based Security-Aware Elasticity

There is a tradeoff between the performance and security requirements. It is useful to use a model-based checking method to operate some runtime decisions for considering a user-defined balance between them. This section expresses a case of the probabilistic Markov decision process (MDP) model. This model can be used to cover both performance and security issues in various NoSQL databases. MDPs are defined by their states, actions, rewards, and transition probabilities (Puterman 2014). Each state belongs to a different cluster size, where the size is equal to the number of active cloud virtual machines (VMs) running an NoSQL database (e.g., Cassandra). The NoSQL database usually uses both of the sharded and replicated types. There exist three types of the possible actions on any state: (i) *add* for VM additions, (ii) *rem* for removals, and (iii) *no_op* for no operation. Figure 14.11 shows a simplified instance of the MDP model, where the states indicate the number of active VMs. The edges represent three possible actions as follows: (i) add_x (blue arrows), (ii) rem_x (red arrows), and (iii) *no_op* (black arrows), where x indicates the number of new or removed VMs. In this instance, the maximum number of VMs in each step is two and the number of active VMs is three. The action types are labeled on top of each transition ($[add_x/rem_x/no_op]$). The MDP relates to a reward value of each state and action taking into the current external conditions. Such conditions can be calculated as the amount of submitted queries per time unit (Naskos et al. 2015).

14.2.3 Security Problems of the NoSQL Environments

Security in the NoSQL databases is not very strong. Furthermore, authentication and encryption procedures are not existence or very weak in the implementation phases. Compared to relational database, NoSQL databases offer a very weak layer of security.

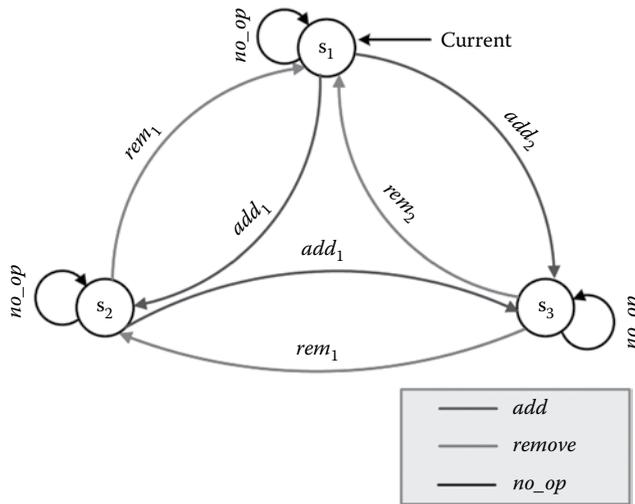


FIGURE 14.11
An overview of the MDP model.

The NoSQL databases are presented with different security issues. They are constructed to meet the requirements of the analytical world associated to big data; moreover, less security issues are considered during the design phase. To solve the security problems of the NoSQL databases, developers have to embed the security strategies at the middle-ware along with the databases. Figure 14.12 depicts some of the major problems in the NoSQL databases (Padhy and Panigrahy 2015).

Denial of Service is one of the problems in the Cassandra database. It is emerged because this database uses a Thread-Per-Client model in the network code. Because the Cassandra server starts a new thread on each TCP connection, it recommends using some sorts of the connection pooling. The new client connections can be prevented by attackers in the Cassandra server. This is done by allocating all its resources to fake the connection attempts. The IP addresses of the cluster members are the only pieces of information required by attackers. This information can be achieved by sniffing the network, passively. The exiting implementation does not time out the passive connections that cause the different threads created in the network (Okman et al. 2011).

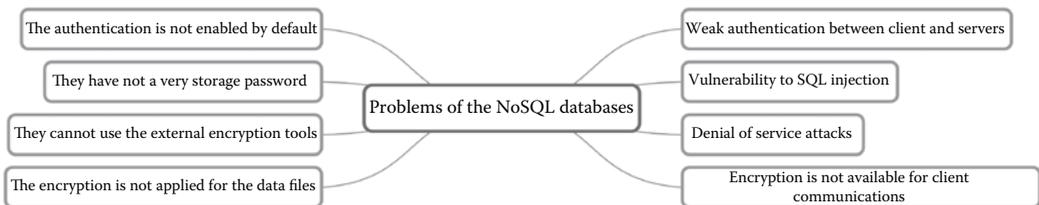


FIGURE 14.12
The major problems of the NoSQL databases.

14.3 Privacy Challenges of NoSQL Databases

Privacy is, specially, relevant as there are new calls to share big data among industry sectors. It means that data must be used only for the purposes for which they were collected by organizations. The ability to extract, analyze, and correlate the sensitive data sets are some of the technological limitations on privacy of the big data. The dominant advances in big data analytics have led to extract and correlate these data to make easily privacy violations. Hence, big data applications should be developed with the knowledge of privacy principles and recommendations. The large-scale collection and data storage make big data attractive to many parties including industry, government, and criminals (Cardenas et al. 2013). Some of the common privacy concerns are shown in [Figure 14.13](#).

Education industry uses NoSQL storage systems to store data of the educational institutions. A high volume of educational information can be maintained in NoSQL databases due to their high ability. Customers of big data are worried about the misuse of these data in the education industry. Instead of managing big data on their own servers, public schools utilize third-party groups, which increases the concern of parents and policy makers. In this area, some of the cloud services are weakly understood, nontransparent, and poorly governed.

The health industry shares some privacy concerns similar to the education industry. Data breaches increase in size and frequency while health organizations transfer their obtained records from old filing cabinets to digital. A hospital in the United States reported that more than 34,000 of their patients annoy due to their compromised medical data. This problem emerged due to a contractor who had downloaded data files from the hospital database and lost his laptop because of theft.

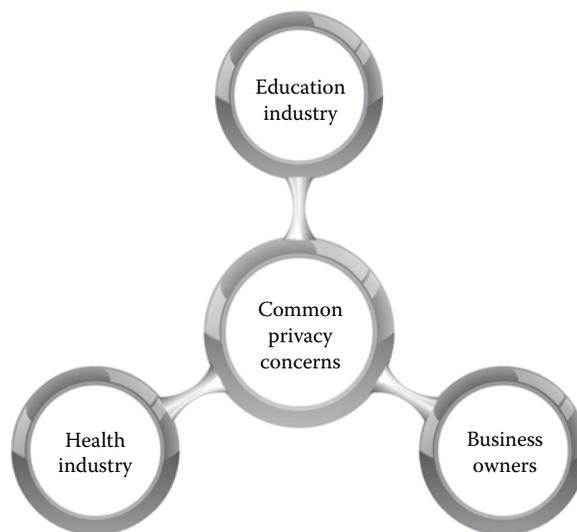


FIGURE 14.13

An overview of the common privacy concerns.

Business owners can define the business requirements (such as SLAs, uptime, and data sources) based on the right technology for business needs. The developers can take ownership of the data model design using NoSQL database systems in business environments. It is possible that business owners find it more difficult to obtain consumer information to share their personal data. They have to satisfy their customers with their experience and the ability of the business. Advertisers may find it difficult to get their targeted publics and acquire information from potential buyers and businesses (Big Data Privacy Concerns and Solutions 2016).

Privacy requirements are specified in the NoSQL database systems to ensure personal controls over big data stored in such databases. That is, a database owner specifies who can see and use information according to their discretion. As represented in [Table 14.5](#), privacy challenges in some of the NoSQL databases are as follows: (i) MongoDB uses the unencrypted data on data stores; (ii) there is not any encryption strategy available for client communications in Cassandra; (iii) there is not any security mechanism at the data level, and data encryption as well as communication with database is not encrypted in the NEO4J database; and (iv) Hadoop/HBASE does not consider encryption mechanisms on the wire (Kadebu and Mapanga 2014).

14.3.1 The Privacy of Data Owners and Query Users

Privacy of data owners and query users is very essential in modern cloud-based data management. It is vital to be considered by researchers to focus on the privacy of data owners or query users. This is a great challenge to the privacy of the data owners and users. A solution of data storage and query protocol is presented in Guo et al. (2013) that works based on classical homomorphic encryption scheme. This protocol preserves privacy of both data owners and query users, simultaneously.

In main database files, data files are stored as the key–value pair. The pairs are provided by NoSQL storage structure and encrypted with Elgamal homomorphic encryption. As shown in [Figure 14.14](#), keys of the index are ciphertext of the composited real keys in big blocks that are encrypted with Paillier encryption scheme and are additive homomorphic cryptosystems. The ciphertext in blocks can be compared to improve efficiency of query when a key is queried by the system. Protocols are presented for data manipulation and query among data owners, service providers, and querying users. To verify usefulness conditions of the solution, algorithms for data updating and querying operations are implemented in this work. Berkley DB, a typical key–value pair model database, is selected to implement the solution.

TABLE 14.5
Privacy Challenges in Some NoSQL Databases

NoSQL Database	Privacy Conditions
MongoDB	Unencrypted data
Cassandra	No encryption available for client communication
NEO4J	Without security at the data level, without data encryption, and communication with unencrypted database
Hadoop/HBASE	Without any encryption on the wire

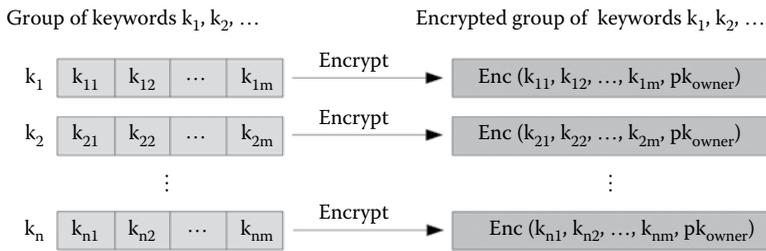


FIGURE 14.14
An overview on the structure of index keys.

14.3.2 Solutions for Existing Privacy Problems

Some of the scalable privacy-preserving data mining and analytics are represented in Mora (2012). Big data is considered as a troubling manifestation by enabling the invasions of privacy, invasive marketing, reduced civil freedoms, and increased corporation control. Already, anonymized data for analytics purposes are not enough to hold user privacy, for example, AOL released the anonymized search logs to perform some of the academic purposes, but users could be easily identified by their searchers. Moreover, Netflix discovered a similar problem while users of their anonymized data set were recognized by correlating their Netflix movie scores with IMDB scores. Hence, it is very essential to present guidelines and recommendations for preventing the inadvertent privacy disclosures. Companies and government agencies collect user data to mine and analyze by inside analysts, outside contractors, or business partners. A malicious insider or untrusted partner can misapply data sets to obtain private information from customers. The cryptographically enforced access control and the secure communications are some of the solutions to the above problem.

Figure 14.15 represents three solutions to address privacy concerns. The first solution is to encrypt hard drives and not to ignore security updates. They seem to be basic ways but more effective to protect both of consumers and enterprise data. The second solution is not to store data on the enterprise servers. Data do not need to be maintained on business servers as well as all applications do not require data storage

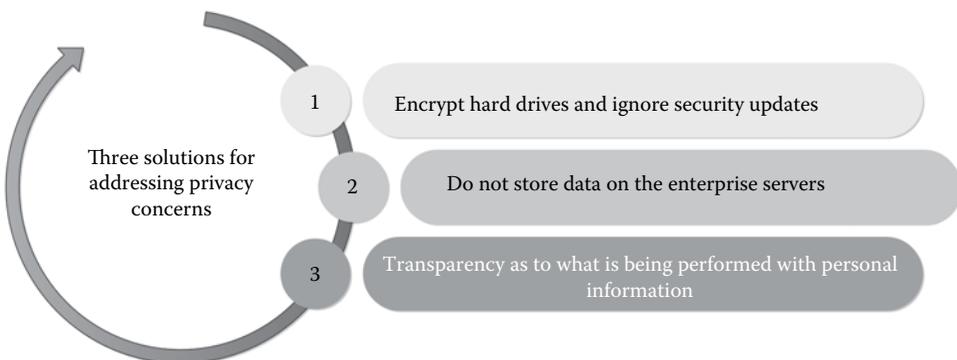


FIGURE 14.15
Some solutions to address privacy concerns.

TABLE 14.6

Four Recommendations for the Internet Users' Privacy on Amazon

Recommendation	Operation Area	Instance on the Amazon Website
1	Registered users' privacy	Wt.II.1, Wt.IV.1, and Wt.IV.5
2	Unregistered users' privacy	Wt.I.1, Wt.III.1, and Wt.III.2
3	Password choice	The area secured by HTTPS
4	Card details confidentiality	The area supported by credit cards

on business servers as proven by some apps such as WhatsApp and Snapchat. The third solution to put organizations in a positive light is to consider transparency as to what is being performed with personal information. This lets consumers to receive the good feeling about they are the main owner of their own data as well as organizations will offer the trust and brand loyalty of their services (Big Data Privacy Concerns and Solutions 2016).

Amazon is one of the world's largest e-shopping sites with a big marketing era. The interaction process between its website and Internet users is an example of a widespread security ceremony. It leads to emerge the security-related interaction of the users with some technologies such as security protocols and online service consumption. A work is represented in Bella and Coles-Kemp (2011) to discuss on how Amazon's ceremony manages the users' privacy through digital identities. It pinpoints some risks affecting the users' privacy. The work presents four recommendations for technical website updates to resolve the mentioned risks. These recommendations address some of the important contexts (e.g., users accessing Amazon from their smart phones) as represented in [Table 14.6](#). Recommendation 1 can protect the network storage involved by personal information of the users' computers in a way that the information is not essentially secured. Recommendation 2 represents that only registered users are allowed to perform searches on the products after they login. Recommendation 3 considers that the Amazon website encourages each user to select a strong password. Recommendation 4 indicates that the Amazon website should grant its users the choice to record their card details to explain the protection measures.

14.4 Comparison Results

The security features of the most popular NoSQL databases are compared to each other as represented in [Table 14.7](#) (Grolinger et al. 2013). The comparison process is performed in terms of encryption, AAA. The summary results indicate that each NoSQL database involves some capabilities to serve the desired requests. In general, the security features of NoSQL databases are not as mature as those involved in relational databases. Many solutions are provided to be only used in secure networked environments. Hence, it is the network administrator's responsibility to ensure that only authorized users and applications can access the data stores. In these cases, there is not any fine-grained access to the data stores. Moreover, audit features, considered by the most cases, are very simple and not customizable. The summary results represent that MongoDB and Cassandra provide additional security capabilities in their enterprise editions.

TABLE 14.7

The Comparison between the Security Features of the Most Popular NoSQL Databases

Database		MongoDB	Cassandra	CouchDB	HBase	HyperTable
Encryption	Data at rest	No, but a third-party partner offers an encryption plug-in	Only in Enterprise Edition. Commit log is not encrypted	NA	No, planned for future release	No
	Client/server	Yes, SSL based	Yes, SSL based	Yes, SSL based	Yes	NA, embedded data store
	Server/server	Yes	Yes, only between datacenters or between servers in the same rack	Possible by HTTPS connections	HBase nodes with the HDFS and Zookeeper clusters can be secured	No
Authentication		Yes, maintain credentials in a system collection	Yes, maintain credentials in a system table	Yes, HTTP authentication by cookies or BASIC method	Yes, RPC API based on SASL, supporting Kerberos	No
Authorization		Yes, permissions contain read, read/write, dbAdmin, and userAdmin	Yes, like the SQL GRANT/REVOKE approach	Complex authorization can be performed in validation functions	Yes, permissions involve read, write, create, and admin	No
Auditing		No	Only in Enterprise Edition	No	No, planned for future release	No

References

- Andreeva, J., Beche, A., Belov, S., Dzhunov, I., Kadochnikov, I., Karavakis, E., Saiz, P., Schovancova, J., and Tuckett, D. 2014. Processing of the WLCG monitoring data using NoSQL. *J. Phys. Conf. Ser.*, 513(3), 032048.
- Bella, G. and Coles-Kemp, L. 2011. Internet users' security and privacy while they interact with Amazon. In *Proc. IEEE 10th Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Changsha, China, pp. 878–883.
- Big Data Privacy Concerns and Solutions. 2016. Retrieved from <http://www.toadworld.com>
- Boicea, A., Radulescu, F., and Agapin, L. I. 2012. MongoDB vs Oracle-database comparison. In *Proc. IEEE 3th Int. Conf. on Emerging Intelligent Data and Web Technologies*, Bucharest, pp. 330–335.
- Cardenas, A. A., Manadhata, P. K., and Rajan, S. P. 2013. Big data analytics for security. *IEEE Secur. Priv.*, 6, 74–76.
- Database security: At rest, but not at risk. 2016. Retrieved from <http://www.csoonline.com>
- Fidelis Cybersecurity. 2014. Current Data Security Issues of NoSQL Databases. Retrieved from <https://www.fidelissecurity.com>

- Grolinger, K., Higashino, W. A., Tiwari, A., and Capretz, M. A. 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *J. Cloud Comp. Adv. Sys. Appl.*, 2, 22.
- Guo, Y., Zhang, L., Lin, F., and Li, X. 2013. A solution for privacy-preserving data manipulation and query on NoSQL database. *J. Comp.*, 8, 1427–1432.
- Han, J., Haihong, E., Le, G., and Du, J. 2011. Survey on NoSQL database. In *Proc. IEEE 6th Int. Conf. on Pervasive Computing and Applications (ICPCA)*, Port Elizabeth, South Africa, pp. 363–366.
- HyperTable. 2016. Retrieved from <http://www.hypertable.com>
- Jeong, S., Zhang, Y., Lynch, J. P., Sohn, H., and LAW, K. H. 2015. A NoSQL-based data management infrastructure for bridge monitoring database. In *Proc. 10th Int. Workshop on Structural Health Monitoring (IWSHM)*, Stanford, pp. 1567–1574, doi:10.12783/SHM2015/196.
- Kadebu, P. and Mapanga, I. 2014. A security requirements perspective towards a secured NOSQL database environment. In *Proc. Int. Conf. Advance Research and Innovation (ICARI)*, New Delhi, India, pp. 472–480.
- Khetrapal, A. and Ganesh, V. 2006. *HBase and Hypertable for Large Scale Distributed Storage Systems*. Department of Computer Science, Purdue University, Indiana, pp. 22–28.
- Lane, A. 2012. Securing big data: Security recommendations for Hadoop and NoSQL environments, *Securosis*, 1, 1–16.
- Maier, D. 1983. *The Theory of Relational Databases*. Rockville, Maryland: Computer Science Press, vol. 11, ISBN: 978-0-914-89442-1.
- Mohamed, M. A., Altrafi, O. G., and Ismail, M. O. 2014. Relational vs. NoSQL databases: A survey. *Int. J. Comput. Inf. Tech.*, 3, 598–601.
- Mora, C. A. 2012. Top ten big data security and privacy challenges. Cloud Security Alliance. Retrieved from https://www.isaca.org/Groups/Professional-English/big-data/GroupDocuments/Big_Data_Top_Ten_v1.pdf
- Naskos, A., Gounaris, A., Mouratidis, H., and Katsaros, P. 2015. Security-aware elasticity for NoSQL databases. In L. Bellatreche and Y. Manolopoulos (eds.), *Model and Data Engineering* (pp. 181–197). Springer, Rhodes, Greece. Doi: 10.1007/978-3-319-23781-7_15.
- NoSQL Vendor Comparison. 2016. Retrieved from <https://www.mongodb.com>
- Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., and Abramov, J. 2011. Security issues in NoSQL databases. In *Proc. IEEE 10th Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Changsha, China, pp. 541–547.
- Padhy, R. P. and Panigrahy, D. 2015. NoSQL databases: State-of-the-art and security challenges. *Int. J. Recent Eng. Sci.*, 16, 38–46.
- Puterman, M. L. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons.
- Rocha, L., Vale, F., Cirilo, E., Barbosa, D., and Mourão, F. 2015. A framework for migrating relational datasets to NoSQL. *Procedia Comput. Sci.*, 51, 2593–2602.
- Zahid, A., Masood, R., and Shibli, M. A. 2014. Security of sharded NoSQL databases: A comparative analysis. In *Proc. IEEE Conf. on Information Assurance and Cyber Security (CIACS)*, Rawalpindi, Pakistan, pp. 1–8.

15

Attack Graph Generation and Analysis Using Graph Database

Mridul Sankar Barik, Anirban Sengupta, and Chandan Mazumdar

CONTENTS

15.1 Introduction	291
15.2 Background on Attack Graph	292
15.2.1 Algorithms/Tools for Attack Graph Generation	293
15.2.2 Algorithms/Tools for Attack Graph Analysis	296
15.2.3 Challenges in Attack Graph Processing	297
15.3 Graph Database	297
15.3.1 Current State of the Art of Graph Databases	298
15.3.2 Graph Query Languages	298
15.3.3 Neo4J and Cypher Query Language.....	299
15.4 Attack Graph and Graph Database	300
15.4.1 Attack Graph Data Model.....	300
15.4.2 Attack Graph Generation.....	303
15.4.3 Attack Graph Analysis	307
References.....	308

15.1 Introduction

With growing dependence on ICT-based systems, securing such systems poses a great challenge. The number and types of attacks against those systems are ever increasing and are a growing concern for security administrators. Attack graph is a useful modeling tool for enumerating multistage, multihost attacks in organizational networks. Without this tool, it is very difficult even for experienced security analysts to manually discover, how an attacker can combine vulnerabilities (in the same host or in connected hosts) to compromise critical resources. An attack graph that shows all possible multistage, multihost attack paths is crucial to a security administrator. It helps in understanding the diverse nature of threats and to decide on appropriate countermeasures. This requires efficient and scalable solution for attack graph generation and its analysis.

Moreover, interactive analysis of attack graph is useful when changes in host/network configuration and/or security objectives occur frequently. Existing approaches (Wang et al. 2006, 2008) use relational model of attack graph to enable security analysts implement custom analysis algorithms on the fly in terms of SQL queries, thereby reducing response time. But, relational databases perform poorly and does not scale well while handling large graph data such as attack graphs, due to lack of data structures and operations related to graphs.

In recent years, there has been a reemergence of interests in graph databases for storing and managing large graph data such as social, web, and biological graphs. A graph database is a kind of NoSQL database that allows persistent storage of entities and relationships between these entities. Entities and relationships represent the set of nodes and edges of a graph, respectively. Relational data modeling follows strict data model and is not suitable for semistructured data. Moreover, SQL, the standard query language for relational databases, cannot express paths which are essential for graph-based reasoning. On the other hand, graph databases use powerful data model suitable for semistructured and connected data. Also, the graph query languages have been specifically designed for graph-based reasoning. For attack graph analysis, graph database is suitable as the analysis tasks in most of the cases involve local processing around a node within a larger global attack graph. Very few analyses are based on global characteristics of attack graph. Hence, use of graph databases in comparison to relational databases immensely increases performance when dealing with connected data.

Neo4J is a popular graph database having native processing capabilities as well as native graph storage. It also has a built-in expressive query language, Cypher. This book chapter discusses a use case scenario of Neo4J for attack graph generation and analysis.

15.2 Background on Attack Graph

Traditionally, defense approaches against adversarial attacks in enterprise networks have been mostly host centric. In this approach, attention is given to identifying vulnerabilities of the individual hosts and taking measures to mitigate them. Vulnerability scanning tools, such as Nessus, OpenVAS, and Nexpose, provide per-host vulnerability information and help in achieving these objectives. However, one major problem with this approach is that it emphasizes more on host-specific local information and does not consider them in the light of global security context of the network. Theoretically, an exhaustive vulnerability searching and patching may lead to a secure system. However, this defense measure may not be possible in practice due to the costs involved and sometimes due to operational constraints. Moreover, in many cases, attackers cleverly combine elementary attacks to launch multistage attacks against critical assets. These elementary attacks exploit vulnerabilities of individual hosts and may be either remote or local. Intrusion detection systems, either network or host based, can detect those elementary attacks but cannot report whether they are part of a larger attack chain or not.

An attack graph is an important modeling tool used in the assessment of security of enterprise networks. Using attack graphs, network administrators can understand how an attacker can combine vulnerabilities in multiple hosts in a multistage attack to compromise critical resources in a network. An exhaustive attack graph of a network provides a global view of its security posture enabling quantitative assessment of the same. Such assessments, when performed periodically, help a network system to evolve overtime.

The concept of attack graph was first introduced in Phillips and Swiler (1998) and Swiler et al. (1998). Since then, it has attracted a lot of attention from researchers, and a considerable amount of research effort has been spent in the development of theory and practices around the idea of attack graph. Initial research proposed custom algorithms, model checking, logic-based, and relational model-based approaches as attack graph generation methods. However, the scalability issue in attack graph generation is still a challenging

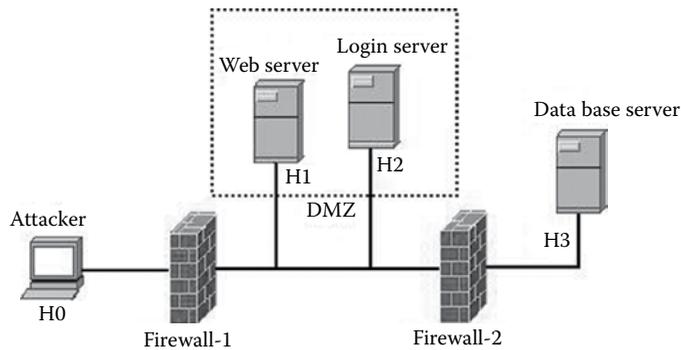


FIGURE 15.1
Simple network configuration.

research problem. Other research efforts aimed at using attack graphs for analyzing or quantifying security risks of enterprise networks.

Figure 15.1 shows a simple network configuration that is used as a running example throughout this chapter. In this network configuration, Firewall-1 controls traffic between the external and internal networks. Assumed location of that attacker is on host H0 in the external network. In the DMZ of internal network, a web server runs on host H1 and a login server (via ssh) on host H2. The web service requires access to a back-end database server which is running on host H3. Firewall-1 allows http and ssh traffic to the web server and login server, respectively, and blocks all other traffic. Firewall-2 allows access to the database server coming only from the web server. Also, a host-based firewall in host H1 prevents access to web server from any host in the DMZ. Host H1 is running a vulnerable version of Apache web server, which has vulnerability (CVE-2006-3747) that allows remote attackers to exploit and gain user privilege on the web server. The SSH service on H2 has a vulnerability (CVE-2002-0640) which allows remote attackers to gain user privilege. Database server H3 is a Linux box running MySQL database which has a remotely exploitable vulnerability (CVE-2009-2446), enabling attacker to gain user privilege. Attacker's objective is to gain user privilege on the database server.

15.2.1 Algorithms/Tools for Attack Graph Generation

In earlier days, dedicated security teams (called Red teams) used to determine overall security of networks by hand-drawing gigantic attack graphs and then analyzing them. Obviously, this approach was tedious, error prone, and did not scale up as the network size grew. This gave rise to the need for automated methods of attack graph generation.

Initial works (Phillips and Swiler 1998; Swiler et al. 1998) on attack graph introduced an attack graph representation known as the state enumeration graph. In this formalism, nodes represent possible system state during execution of an attack. A system state typically comprises information on host(s), user access levels, and effects of the attack so far. Edges represent a change of state caused by a single action of the attacker and may be weighted based on the attacker effort required or the time to succeed. Based on this formalism, the algorithm presented in Swiler et al. (2001) generates attack graph automatically starting from the initial state. It matches attack templates to the configuration of the network system and attacker's profile in a forward exploration manner and generates the

graph iteratively. To put it another way, the attack graph is an instantiation of the attack templates to the configuration information and attacker profile.

Ritchey and Ammann (2000) proposed the use of model-checking techniques for attack graph generation. One benefit of using model-checking techniques over custom algorithms for attack graph generation is that users need not worry about the problem of handling large state space, which is otherwise elegantly handled by standard model-checking software. In this approach, desirable security properties of the system are specified as logic formulae, and the model checker finds out whether the model of the system satisfies the formulae. If not, it generates counterexamples that show the sequence of states, from the initial state to the state where the property is violated. As each state change corresponds to the event of execution of an exploit, a counterexample produces an attack path that describes the sequence of exploits that leads an attacker from its initial position to a compromised resource. A set of all such counterexamples would produce a complete attack graph and has been termed as a scenario graph (Sheyner et al. 2002).

The approaches based on the state enumeration graph or the scenario graph suffered from scalability issues due to full exponential state space. The monotonicity assumption on attacker's behavior, first introduced in Ammann et al. (2002), was a key enabler in handling this issue. This assumption says that preconditions of an attack are never invalidated by successful execution of another attack. The resulting graph, which enumerates all such possible exploit sequences, is known as the exploit dependency graph.

The logic programming-based approach to network security analysis known as Multihost, Multistage Vulnerability AnaLysis (MuIVAL) was introduced in Ou et al. (2006). It adopts a representation of attack graph known as logical attack graph which shows logical dependencies among attack goals and configuration information. A node in the logical attack graph is a logical statement that encodes only some part of network state. Unlike the state enumeration graph or scenario graph, it does not represent or encode the entire state of the network. Edges represent the causality relationships between various network configurations.

The Network Security Planning Architecture (NetSPA) (Ingols et al. 2006) attack graph generation system is based on a new representation of attack graph, that is, the multiple prerequisite graph. This tool uses readily available source of data to automatically compute network reachability, classify vulnerabilities, build the graph, and recommend actions to improve network security. A multiple-prerequisite attack graph consists of three types of nodes, that is, state, prerequisite, and vulnerability instance nodes. State nodes represent attacker's level of access on a given host. Prerequisite nodes represent either a reachability group or preconditions of one or several attacks. Vulnerability instance nodes represent particular vulnerabilities. Directed edges from state nodes to prerequisite nodes represent the capabilities those states enable for the attacker. Prerequisite nodes point to vulnerability instance nodes that represent the set of attacks that the prerequisite node enables. Directed edge from vulnerability instance nodes to a single-state node represents the state the attacker can reach by successfully exploiting the vulnerability.

Topological vulnerability analysis (TVA) (Jajodia et al. 2005; Jajodia and Noel 2007) adopts a topological approach to network vulnerability analysis. It considers a set of modeled attacker exploits on a network and then finds out different sequences of exploits or attack paths starting from attacker's initial state leading to compromise of critical network assets. For this, TVA requires an extensive knowledgebase of known vulnerabilities and attack techniques. TVA attack graph generation engine uses the algorithm proposed in Ammann et al. (2002) for attack graph generation.

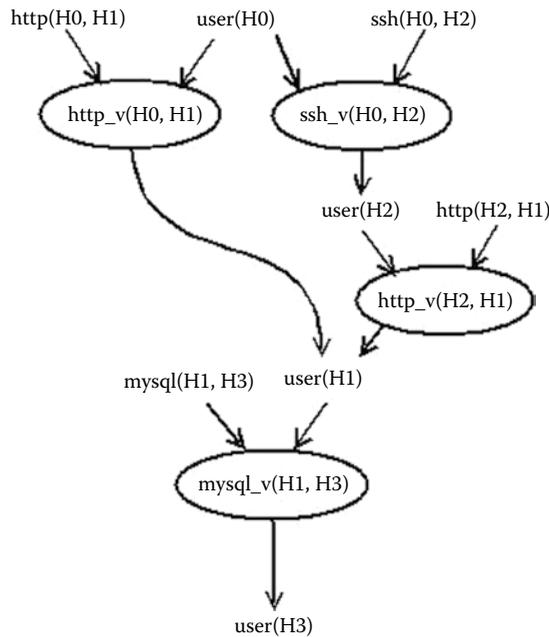


FIGURE 15.2
Exploit dependency attack graph.

The attack graph model used in TVA uses two types of nodes: exploit nodes and security condition nodes. This model of attack graph is based on exploit dependency graphs. Exploit nodes represent attacks (exploitation of certain vulnerabilities) and security condition nodes represent either the attack postcondition or precondition. An exploit is defined by pre- and postconditions. Directed edges from security condition nodes to attack nodes represent preconditions of an attack, of which all must be met for an attack to be successful. A directed edge from an attack node to a security condition node represents postcondition of an attack. An advantage of exploit dependency graphs is that instead of modeling hosts, exploits on hosts are modeled, thus reducing the computational complexity. On the other hand, this model requires information on low-level attack details. Vulnerability information are based on pre- and postconditions. [Figure 15.2](#) shows the exploit dependency attack graph for the example network of [Figure 15.1](#).

In exploit dependency attack graph, ovals represent exploits and are labeled with corresponding vulnerabilities. Other nodes represent either some network condition or attacker capability. For example, network condition $\text{http}(H0, H1)$ means accessibility of web service on host $H1$ from host $H0$. Attacker capability $\text{user}(H0)$ means, attacker has user privilege on host $H0$. Directed edges in and out of exploit nodes identify pre- and postconditions, respectively, of an attack. For example, exploitation of MySQL vulnerability CVE-2009-2446 on host $H3$ from host $H1$, that is, $V3(H1, H3)$ requires preconditions $\text{user}(H1)$ and $\text{mysql}(H1, H3)$ and generates a postcondition $\text{user}(H3)$. Exploit dependency attack graph elegantly enumerates different attack paths leading to a critical resource. In this example, there are two attack paths leading to attacker gaining user privilege on $H3$. They are $\text{http_v}(H0, H1) \rightarrow \text{mysql_v}(H1, H3)$ and $\text{ssh_v}(H0, H2) \rightarrow \text{http_v}(H2, H1) \rightarrow \text{mysql_v}(H1, H3)$, respectively.

15.2.2 Algorithms/Tools for Attack Graph Analysis

The sole objective of generating an attack graph is to enable assessment of security. There are many ways in which information encoded in an attack graph can be used to gain vital insight into the global security posture of a network. This also helps security administrators to make correct decisions about mitigation strategies. Different attack graph-based analysis techniques have been proposed in the literature; some of them are discussed briefly below:

- *IDS alert correlation and sensor placement.* Multistep network intrusions comprise multiple attack steps with one preparing for the next. Intrusion detection system alert correlation techniques help in deciding whether an isolated alert is part of an ongoing multistep network intrusion. It also helps in attack scenario reconstruction. Noel et al. (2004) first reported the use of attack graphs in minimizing the effect of false alarms by correlating isolated intrusion alerts as part of multistep attack paths. Their alert correlation method is based on the shortest distance between exploits in the attack graph. Also, any IDS alert that does not feature in the possible future activities of the attacker (as can be observed in the attack graph) can readily be classified as false. TVA attack graph has been used for planning optimal placement of IDS sensors (Noel and Jajodia 2008) against all possible attacks. In this technique, isolated intrusion alerts are mapped to known exploits (represented as nodes) in an attack graph. It enables correlation of alerts corresponding to a multistep attack scenario and also prioritization of alerts based on the distance from critical network assets. Furthermore, using the knowledge of attack paths encoded in an attack graph, network administrators can formulate the best possible options for responding to different attacks.
- *Minimum cost network hardening.* An attack graph reveals the different ways network resources can be compromised, but it does not provide any direct solution to harden the network. One of the network-hardening measures is to remove or patch vulnerabilities. A good network-hardening approach should remove those vulnerabilities such that none of the attack paths leading to given critical resources can be realized, and also the cost involved in removing those vulnerabilities is minimum (Noel et al. 2003; Wang et al. 2006).
- *Network forensics.* Forensic analysis is typically performed after an incidence of break-in occurs. Its objective is to find attacker's probable actions, to assess damage, and to collect digital evidence in case legal action is required. Attack graph-based forensic analysis enables administrators to prove that a series of IDS alerts are not isolated; rather, they correspond to a sequence of attacks in a coherent attack plan. Liu et al. (2012) proposed a solution where they have augmented attack graphs with antiforensic activity nodes that help in identifying missing evidences.
- *Attack graph-based security metrics.* To improve the security of a network, administrators must be able to measure it. A security metric measures or assesses the extent to which a system meets its security objectives. Using suitable security metrics, one can measure how secure a network currently is, and how secure it would be after introducing new security mechanisms or configuration changes. This is necessary if a network has to evolve through network hardening. A number of security metrics have been proposed in the literature based on attack graphs.

Network compromise percentage (NCP) security metric indicates the percentage of network hosts where the attacker has obtained user or superuser privilege.

Asset values can be associated with individual hosts before computing the NCP metric. This metric was proposed by Lippmann et al. (2006). Computation of this metric may require traversal of the entire attack graph.

Noel and Jajodia (2014) proposed a suite of security metrics based on attack graph of a network, for measuring overall security risk. The metrics are grouped into families which are then combined into a single score. Single risk score is often beneficial for network administrators in situations where they have to interpret multiple scores. The different families of security metrics are as follows: (i) victimization: scores network services and their vulnerabilities, (ii) size: measures risk in terms of the attack graph size, (iii) containment: measures security risk in terms of the degree with which the attack graph contains attacks across different network protection domains such as different subnets, and (iv) topology: based on graph theoretic properties of the attack graph such as the weakly connected components, strongly connected components, and length of the maximum shortest path.

15.2.3 Challenges in Attack Graph Processing

Since its introduction in 1998, researchers have proposed a variety of solutions for attack graph generation and analysis. Performance results lack evidences of scalability of those solutions for large enterprise networks.

Moreover, most of the existing approaches of attack graph analysis use proprietary algorithms and do not necessarily always use standard graph-based algorithms only. In many situations, such analysis techniques may need to be adapted frequently due to changes in host/network configuration and/or security objectives. Thus, there is always a need for a solution that allows interactive analysis of attack graph. One of the existing works in this area has used relational model for representing network configurations/domain knowledge and used SQL queries to generate attack graph and also to perform typical attack graph analysis operations. The objective of such a solution is to enable security analysts to implement custom analysis algorithms on the fly in terms of relational queries, thereby reducing delay in providing responses to dynamic network conditions. But, it is now a well-accepted fact that for exploration of large graph data such as attack graphs, relational database is not the most practical and scalable solution. This is mainly due to lack of data structures and operations related to graphs in relational databases.

15.3 Graph Database

The proliferation of ubiquitous computation and communication has led to generation of a huge amount of data, further necessitating alternate systems for efficient storage, retrieval, and management. Traditional relational databases face tough challenge while handling a large volume of connected data in application domains like social network analysis, bioinformatics, etc. Implementing such problems in relational databases involves a large number of joins which is costly to be calculated. Also, relational databases are not helpful when the data model evolves overtime, which means that relational databases depend on stiff schema and make it complicated to add new relationships between objects.

Over the past few years, the rise of NoSQL databases has been challenging the dominance of relational databases. The term NoSQL generally has been used to identify database

management systems which do not support SQL queries. The techniques used for data storage and access in NoSQL database are different from relational database management systems. Presently, there are many NoSQL databases available. Each of them has been developed keeping in mind suitability for different application domains. NoSQL databases are generally divided into four classes based on their data model and storage: key–value stores, document stores, column stores, and graph databases. Among them, the graph databases are particularly suitable for applications which deal with a huge amount of connected data.

15.3.1 Current State of the Art of Graph Databases

A graph database management system is a kind of NoSQL database that allows persistent storage of entities and relationships between these entities. It incorporates create, read, update, and delete methods that expose a graph data model. Entities and relationships represent the set of vertices and edges of a graph, respectively.

Some graph databases use *native graph storage* which is optimized and designed for storing and managing graphs. Not all graph databases use native graph storage. Some serialize the graph data into a relational database, object-oriented databases, or other types of general-purpose data stores. Many graph databases have *native graph processing* capability by having a storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary. The neighbors of an entity are accessible by dereferencing a physical pointer. Native graph processing (index-free adjacency) improves graph traversal performance.

Many of the applications handling large graphs not only reason about the graph structure, but also they need to query graph data based on properties of nodes and edges. For example, in a social network, nodes may represent people differing in attribute values such as names and ages. Graph databases supporting property graph model (Property Graph Model 2006) are suitable for this kind of applications. In a property graph, each node and edge is uniquely identifiable. It allows nodes and edges to have a collection of key–value pairs called properties (or attributes). Edge labels describe the type of relationship it represents.

Existing relational data modeling follows strict data model and is not suitable for semistructured data. Moreover, SQL, the standard query language for relational databases, cannot express paths that are essential for graph-based reasoning. On the other hand, graph databases use powerful data model suitable for semistructured and connected data. Also, the graph query languages have been specifically designed for graph-based reasoning. In comparison to the index-intensive, set theoretic operations of relational databases, graph databases make use of index-free traversals. Most of the graph databases use native graph storage and native graph processing engines that leverage index-free adjacency. In such graph databases, called the native graph databases, relationships attached to a node naturally provide a direct connection to other related nodes of interest. Graph queries, as a result, mostly involve using this locality to traverse through the graph, in contrast to joining data through a global index, which are many orders of magnitude slower in the case of RDBMS.

Some of the popular graph databases are Neo4J, HyperGraphDB, Titan InfiniteGraph, DEX, AllegroGraph, Oracle NoSQL Database, etc.

15.3.2 Graph Query Languages

Graph databases provide persistent storage of graphs. Graph query languages facilitate querying and manipulation of graphs with heterogeneous attributes and structures.

There are many graph query languages available with graph database products. SPARQL, Gremlin, and Cypher are the most popular among them. There are also different domain-specific graph query languages which take care of requirements of specific application domains. Examples are SoQL, SNQL, and SocialLite for management and querying of social network data, and Program Query Language (PQL) for specification of application errors and security flaws.

The Resource Description Framework (RDF) is a W3C standard data model to represent information about web resources. SPARQL (SPARQL Query Language for RDF 2016) is syntactically a SQL-like query language for retrieving and manipulating data stored in RDF format. SPARQL also has been adopted as a W3C standard. It is both a protocol and a query language. The SPARQL protocol is meant for remote invocation of SPARQL queries. It describes a simple interface that can be supported via HTTP or SOAP. Client applications can use it to issue SPARQL queries against some endpoint. Some of the features of SPARQL include basic conjunctive patterns, value filters, optional patterns, and pattern disjunction.

Gremlin Query Language (2016) is a graph query language for traversing property graphs and is integrated with Titan graph database. Gremlin also works with most graph databases that support property graph data model. Gremlin can be easily used with JVM languages like Groovy, Clojure, Scala, and more.

Detail of Cypher, the query language of Neo4J, is given in the next section.

Unlike RDBMSs, graph database communities lack a standard query language. More recently, The openCypher Project (2016) aims at standardizing the Cypher query language across different graph data stores. Many of the popular graph database vendors are part of this initiative.

15.3.3 Neo4J and Cypher Query Language

Neo4j (Neo4J 2016) is an open-source NoSQL graph database implemented in Java and Scala. Neo4j implements the property graph model down to the storage level. In comparison to other in-memory graph processing libraries, Neo4j provides full database characteristics including ACID transaction compliance, cluster support, and runtime failover, etc. Neo4J comes in two flavors. The free and open-source Community edition is a high performance, fully ACID-transactional database. The Enterprise editions provide all of the functionality of the Community edition in addition to scalable clustering, failover, high availability, live backups, and comprehensive monitoring.

Cypher is a declarative graph query language for Neo4j graph database. It is the most successful and widely adopted graph query language. Being a declarative language, Cypher focuses on expressing what to retrieve from a graph rather than how to do it. That is, it allows expressive and efficient querying and updating of the graph store without having to write traversals through the graph structure explicitly. Cypher is built around established practices for expressive querying. Among its many features, Cypher allows matching of patterns of nodes and relationships in a graph, to extract information or modify the data. It has the concept of identifiers which denote named, bound elements and parameters. Like SQL, Cypher can create, update, and remove nodes, relationships, and properties from graph database. Following are some clauses used in Cypher query language:

- **START:** Specifies one or more starting points (nodes or relationships) in the graph. These starting points are generally determined via index searches or directly based on node or relationship IDs.
- **MATCH:** The graph pattern to match.

- WHERE: Specifies filtering criteria for pattern matching results.
- RETURN: Specifies which nodes, relationships, and properties in the matched data should be returned.
- CREATE and CREATE UNIQUE: Creates nodes and relationships.
- DELETE: Removes nodes, relationships, and properties.
- REMOVE: Removes properties from nodes or relationships and labels from nodes.
- SET: Set property values.
- WITH: Divides a query into multiple, distinct parts. Chains subsequent query parts and forwards results from one to the next.
- UNION: Merges results from two or more queries.

Neo4J also supports Core API and Traversal Framework API, which allows programmers to fine-tune their algorithms which cannot be expressed effectively using Cypher queries.

15.4 Attack Graph and Graph Database

As discussed earlier, processing attack graph for large enterprise network, including both generation and analysis, is still a challenging task. Existing approaches based on custom algorithms, logic programming, and model checking suffer from scalability issues. Although RDBMS-based approach is simple and allows interactive analysis of attack graph, it suffers from well-known weakness of RDBMS while handling graph data. Rapid advancement in NoSQL database technologies, especially of the graph databases, has resulted in its adoption in a variety of application domains. Although in a nascent stage, attack graph processing is one of them. For attack graph analysis, graph databases are suitable as the analysis tasks in most of the cases involve local processing around a node within a larger global attack graph. Very few analyses are based on global characteristics of attack graph. Till now, very few researchers (Barik and Mazumdar 2014; Noel et al. 2015) have used graph databases for handling large-scale attack graphs. Following subsections describe the attack graph data model, attack graph generation, and analysis using graph queries using Cypher query language.

15.4.1 Attack Graph Data Model

This graph data model describes the entities and relationships that exist between those entities in a problem domain. For attack graph generation, two types of information are necessary. They are (a) network configuration information and (b) domain knowledge. The network configuration information includes (i) network topology information, (ii) firewall (perimeter- and/or host-based) rule set, and (iii) per-host vulnerability information. The domain knowledge refers to the interdependency between different types of vulnerabilities and network conditions, that is, exploit pre- and postconditions.

At first, network configuration information is stored as graph data, and the domain knowledge is encoded as graph patterns. Graph queries are then used to look for existence of such patterns over the graph data representing network configuration

information. Results of those queries provide information about vulnerabilities that can be exploited based on the present network configuration information. The exploitation of such vulnerabilities may generate new network conditions which are also encoded as graph patterns, that is, a set of new nodes and edges. These are added to the existing graph data.

The exploit dependency attack graph representation has been used in this approach. An attack graph in this representation is a directed graph and has two types of nodes: exploits and security conditions. Exploit nodes represent host-bound vulnerabilities and are labeled with triples $\langle v, s, d \rangle$, meaning that vulnerability v on destination host d is exploited from source host s . Security condition nodes represent state of the network that is required for or implied by successful execution of an exploit. Examples of security conditions are access level at a host, accessibility of a service, trusts, etc.

Formally stated, let E and C are the sets of exploits and security conditions, respectively. Then, the *require* relationship R_r relates security conditions to exploits, that is, $R_r \subseteq C \times E$, and the *imply* relationship R_i relates exploits to security conditions, that is, $R_i \subseteq E \times C$. An attack graph is directed graph G with a set of nodes $E \cup C$ and a set of edges $R_r \cup R_i$. Two important aspects of this attack graph representation are as follows. First, the require relation is always conjunctive, whereas the imply relation is always disjunctive. More specifically, an exploit cannot be realized until all of its required conditions have been satisfied, whereas a condition can be satisfied by any one of the realized exploits. Second, the conditions are further classified into initial conditions (the conditions not implied by any exploit) and intermediate conditions. An initial condition can be independently disabled to harden a network, whereas an intermediate condition usually cannot be.

The graph model, as shown in [Figure 15.3](#), consists of a set of nodes V and a set of edges E . There are two types of nodes: entity nodes V_E and fact nodes V_F . Edges represent relationships between entities or between entities and facts.

The basic entities in this graph data model are hosts (H), zones (Z), gateways (GW), services (S), vulnerabilities (V), attacker (A), and attacker goals (G). Hosts reside in a zone, that is, DMZ, external zone, etc. Association of a host in a zone is indicated by an *IN* edge from the host to the zone. Different zones are connected by gateway devices such as routers and firewalls. *CONNECT* edges connect a gateway node with the zones it connects. Each entity has a number of properties/attributes as key-value pairs which uniquely identify those entities.

The fact nodes model interaction among entity nodes. For example, a service instance (*SI*) fact node represents the fact that a given service S is running at a given host H . That *SI* node is connected to the corresponding service node S via *INSTANCE_OF* relationship and to a host node H via *AT_HOST* relationship. A service access instance (*SAI*) fact node represents the fact that a given host H can access a service instance *SI*. That *SAI* node is connected to the service instance node *SI* via *ACCESS_TO* relation and to a host node H via *ACCESS_BY* relation. A *SAI* node enables capturing of information on whether from a source host a service running on a destination host is accessible. The host relation (*HR*) nodes capture directed relations that hold between any two hosts. One *HR* node is connected to the source host node of the relation via a *FROM* relationship and to the destination node via a *TO* relationship. One example of such a relation is "trust." When a host $H1$ trusts $H2$, any user on $H2$ can execute shell commands remotely on $H1$ without providing a password.

Goal instance (*GI*) fact nodes represent the fact about any goal type G which the attacker has gained at a host H . A *GI* node is connected to a goal type node via *INSTANCE_OF*

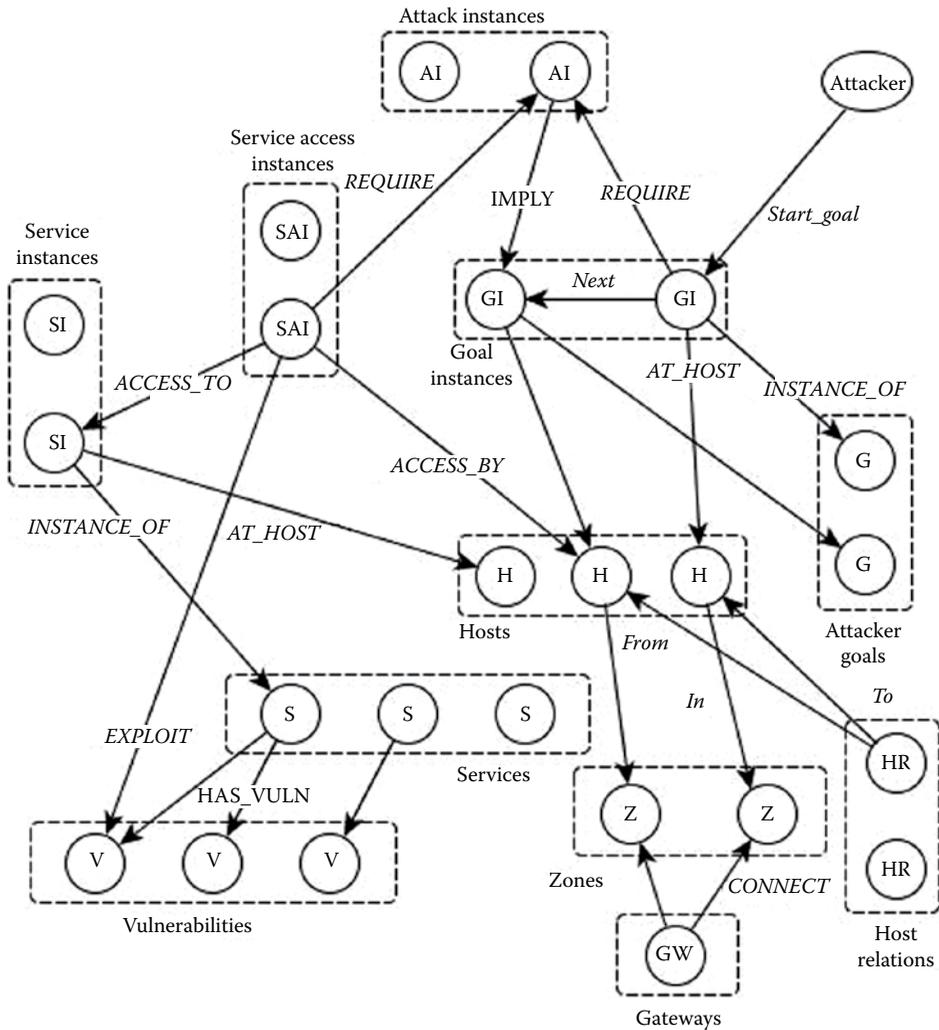


FIGURE 15.3 Attack graph data model.

relationship and to a host node via *AT_HOST* relationship. The attacker must have some initial privileges in the form of GI(s) satisfied at any host. The attacker node is connected to such GIs via *START_GOAL* relationship, to identify attacker’s initial location and privileges. The GIs together with SAIs and HRs act as preconditions that enable the attacker to launch attacks by exploiting vulnerabilities of accessible services. Exploitation of vulnerability is represented by an attack instance (AI) fact node. The AI fact nodes are similar in concept to the exploit nodes in the exploit dependency attack graph representation discussed in Section 15.2.1. The GI facts represent attacker’s capability and the SAI and HR facts represent network conditions. The preconditions of an AI are connected to the AI node via *REQUIRE* relationships. Successful execution of an attack can have two effects. It can either enable a new attacker privilege (in the form of some attacker goals achieved at some host, i.e., new GIs) or it can generate some new SAIs or HRs. The AI node is connected

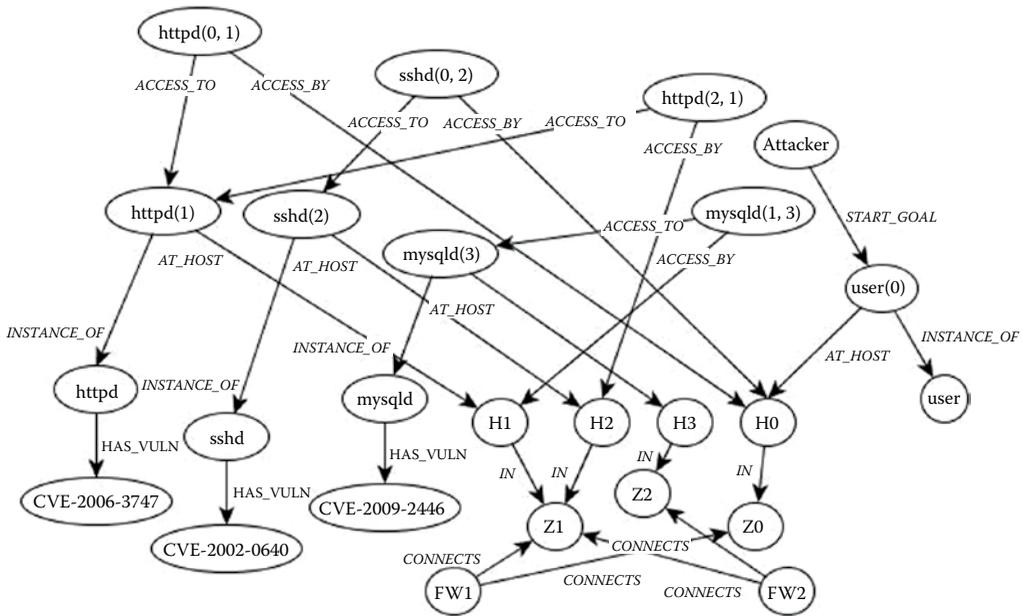


FIGURE 15.4 Graph data of simple network configuration.

to those new nodes via *IMPLY* relationships. Also, each GI node pair (m, n) , which appear as one of the precondition (via *REQUIRE* relationship) and postcondition (via *IMPLY* relationship), respectively, of an AI node, are connected via a directed relationship *NEXT* from m to n . This relationship captures the temporal order among GIs that the attacker achieves in a chain of exploits.

Figure 15.4 shows the graph data corresponding to the example network of Figure 15.1. Here, different types of entity nodes are as follows: services (httpd, sshd, mysqld), vulnerability (CVE-2006-3747, CVE-2002-0640, CVE-2009-2446), host (H0, H1, H2, H3), attacker (Attacker), attacker goal (user), zone (Z0, Z1, Z2), and gateway (FW1, FW2).

Examples of fact nodes are as follows: SIs (http(1), sshd(2), mysqld(3)) and SAIs (httpd(1, 0), sshd(2, 0), httpd(1, 2), mysqld(3, 1)). Note that, these SAI nodes are created after considering all firewall rules (both perimeter and host based).

15.4.2 Attack Graph Generation

The attack graph generation methodology uses an attack pattern-based approach. It builds the attack graph in an iterative manner. In each iteration, all accessible services from the present location of the attacker are found out, such that the services have vulnerabilities which have not yet been exploited.

These exploitable vulnerabilities are then mapped to corresponding attack patterns, and the associated checks are performed. These checking actions involve determining whether all the preconditions for successful exploitation of the concerned vulnerability are satisfied or not and if so, create postconditions as implied. A mapping function AP maps a given vulnerability v , to its corresponding attack pattern $AP(v)$. The set of

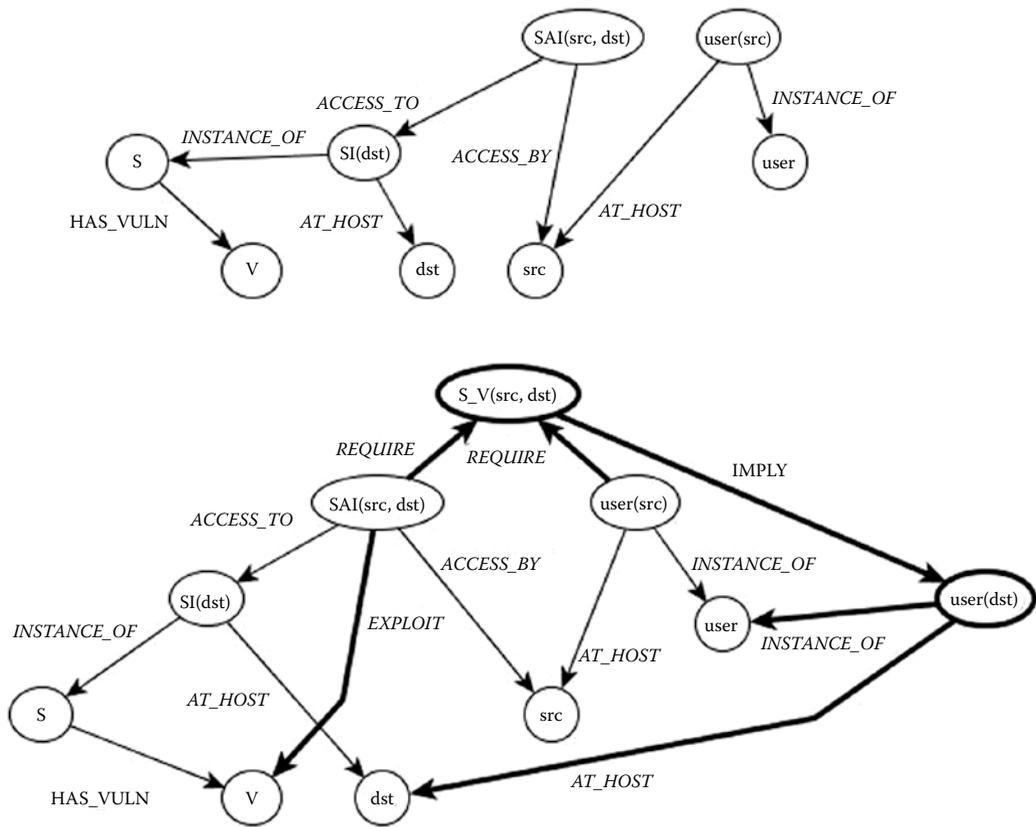


FIGURE 15.5
A simple attack pattern and its corresponding action.

preconditions for an attack pattern (*ap*) is denoted by *ap.precond* and the set of postconditions is denoted by *ap.postcond*.

A simple attack pattern and its corresponding action are shown in Figure 15.5. The attack pattern describes an attack from source host *src* to a destination host *dst*. The preconditions of this attack are as follows: (i) the attacker must have user privilege at *src*, (ii) attacker has access to a service running in *dst* from *src*, and (iii) the service has a vulnerability that is not yet exploited (indicated by the absence of an *EXPLOIT* edge from the corresponding *SAI* node). The action corresponding to this attack pattern actually describes the postconditions of the attack and constructs the attack graph (shown in bold lines). The action in this case involves creation of an exploit node and incoming *REQUIRE* edges from the preconditions of the attack and outgoing *IMPLY* edges to postcondition nodes. It also creates an *EXPLOIT* edge from the corresponding *SAI* node. This prevents the vulnerability to be exploited from the same source host in subsequent iterations.

Given the graph data of simple network configuration of Figure 15.1, the task of finding exploitable vulnerabilities from the present location of attacker can be accomplished very easily using queries in Cypher graph query language. We assume that an attacker goal node has a property “type” which states the kind of goal it is. Following are some example queries in Cypher to achieve different subtasks in attack graph generation:

1. Find source hosts *src* where the attacker has user/superuser privilege.

```
MATCH (src)<-[:AT_HOST]-(gi)-[:INSTANCE_OF]->(g)
WHERE g.type="user" OR g.type="superuser"
RETURN src
```

2. Find all services *s*, running at destination hosts *dst* such that *s* is accessible from the source host *src*.

```
MATCH (src)<-[:ACCESS_BY]-(sai)
RETURN src, sai
```

3. Find any vulnerability *v* of the service *s* not yet exploited from the source host *src*.

```
MATCH (sai)-[:ACCESS_TO]->(si)-[:INSTANCE_OF]->(s)
-[:HAS_VULN]->(v), (si)-[:AT_HOST]->(dst), NOT (sai)-
[:EXPLOIT]->(v)
RETURN s, v, dst
```

The above queries are essentially graph pattern queries over the underlying graph data. When executed on the graph data of our running example, results of these queries show that the attacker from its initial location at H0 can access the httpd and sshd SIs running at H1 and H2, and both instances have unexploited vulnerabilities CVE-2006-3747 and CVE-2002-0640, respectively. Figure 15.6 shows the result.

Next, the attack patterns associated with these two vulnerabilities are instantiated. Both of these attack patterns do not require any extra activities as the sets of preconditions

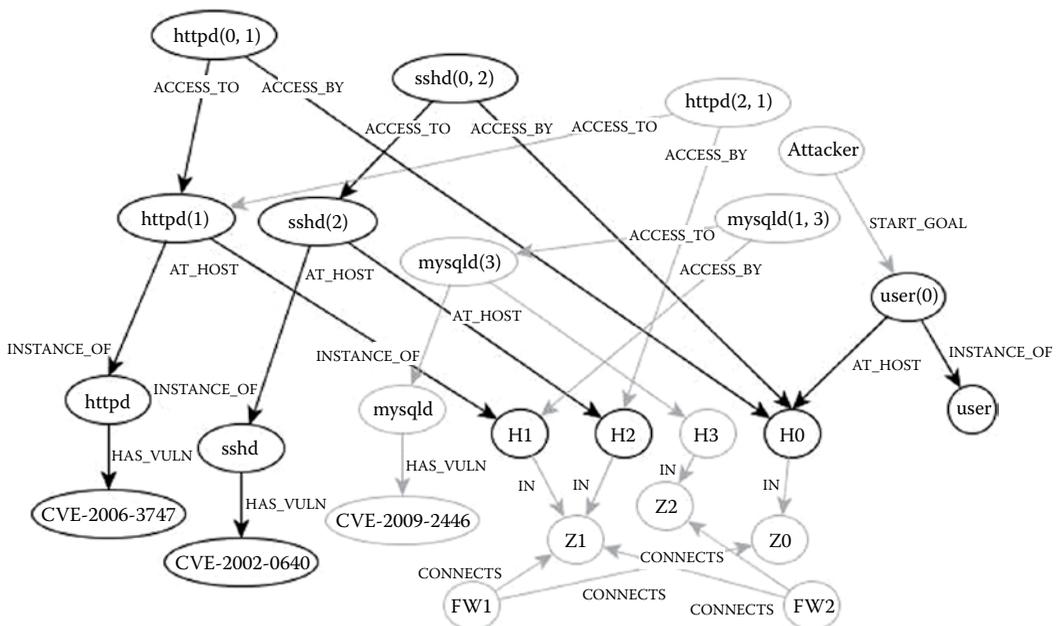


FIGURE 15.6
Finding exploitable vulnerabilities.

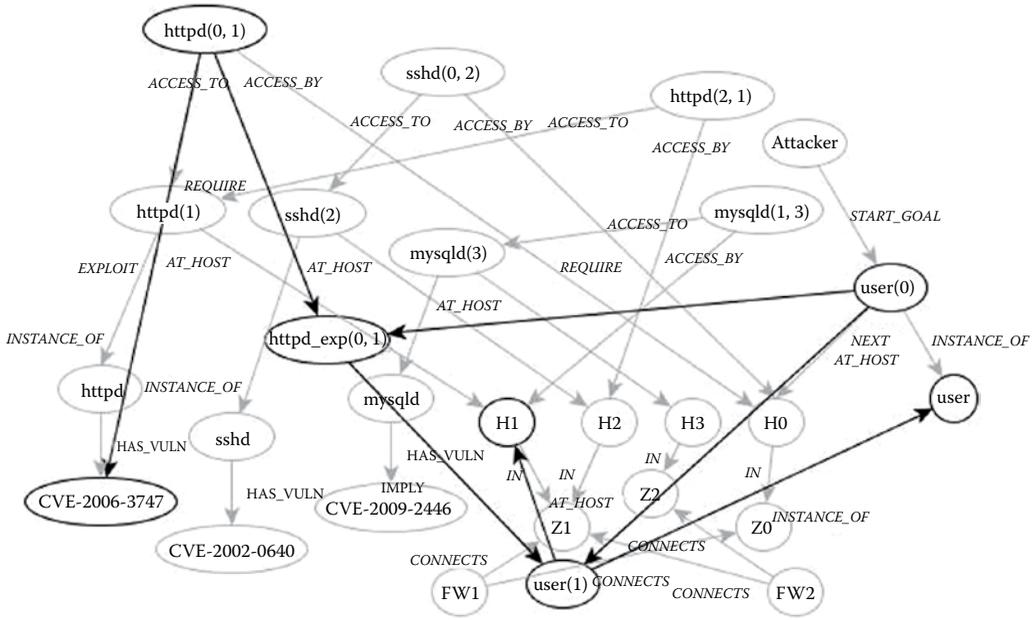


FIGURE 15.7 Postconditions generation after exploitation of web server vulnerability from H0 to H1.

necessary for exploiting the two vulnerabilities, that is, $\{httpd(src, dst), user(src)\}$ and $\{sshd(src, dst), user(src)\}$, are already part of the activity for finding out the accessibility of the vulnerable service itself. The associated attack pattern, however, creates new nodes and relationships as indicated in the template.

When all the preconditions for exploitation of a vulnerability exists, the corresponding postconditions are generated. Figure 15.7 shows the new nodes and edges that are created as a result of exploitation of vulnerability CVE-2006-3747 of httpd service running at host H1 from host H0. The steps involved are as follows: (i) create a new AI node $httpd_exp(0, 1)$, (ii) create *REQUIRE* edges from all precondition nodes $\{httpd(0, 1), user(0)\}$ to the new AI node $httpd_exp(0, 1)$, (iii) create *IMPLY* relationships from the new AI node $httpd_exp(0, 1)$ to all postconditions $\{user(1)\}$, (iv) create *NEXT* relationships from all precondition nodes to postcondition nodes of this vulnerability such that they are of GI type nodes, and (v) create *EXPLOIT* relationship from the SAI $httpd(0, 1)$ node (which allowed attacker at H0, access to service s at H1) to the vulnerability node v . All these activities can be easily achieved using Cypher queries.

In similar manner, postconditions are generated for all exploitable vulnerabilities in the present iteration. Postconditions thus generated may enable exploitation of other vulnerabilities, which are then processed in the next iteration. The generation process stops when exploitable vulnerabilities are found.

Figure 15.8 shows the full attack graph of our running example. The generated attack graph is stored in the same graph database, with direct links to input information for each exploit, that is, AI (although it is not shown in the figure for clarity in presentation).

This approach of attack graph generation is advantageous compared to approaches based on relational databases where large table joins are necessary for determining network conditions that enable an attacker to execute exploits.

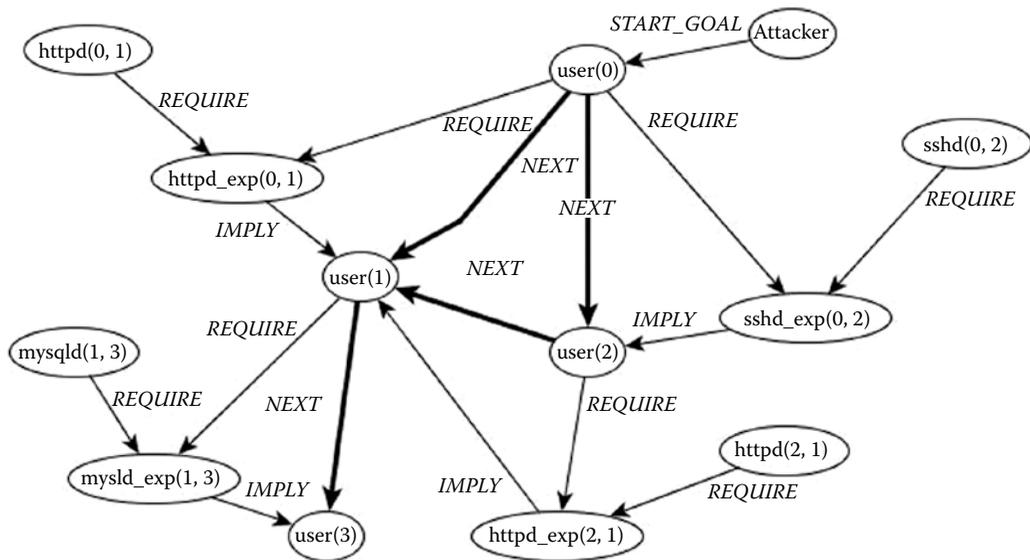


FIGURE 15.8
Full attack graph of example network.

15.4.3 Attack Graph Analysis

One key benefit of using graph databases for representing attack graph is that it enables security administrators to write custom analysis tasks on the fly using graph query language. Analysis techniques based on proprietary algorithms does not provide the needed flexibility. The Cypher graph query language has rich support for path queries. Most of the attack graph-based analysis tasks discussed in Section 15.2.2 can be realized using Cypher path queries. Some examples are given below:

1. Find any attack path (if exists) from a given source host *src* (IP address 1.1.1.1) to a destination *dst* (IP address 2.2.2.2). We assume that each host is uniquely identified by the attribute *ip_addr*.

```
MATCH (src:host{ip_addr="1.1.1.1"})<-[:AT_HOST]-(g1)-
[:INSTANCE_OF]->(g:goal{name:"user"}),
(dst:host{ip_addr="2.2.2.2"})<-[:AT_HOST]-(g2)-
[:INSTANCE_OF]->(g:goal{name:"user"})
WITH g1, g2 MATCH
p=(g1)-[:NEXT*..]->(g2)
RETURN p LIMIT 1
```

2. Find all paths of exploitable vulnerabilities between a particular pair of machines.

```
MATCH (src:host{ip_addr="1.1.1.1"})<-[:AT_HOST]-(g1)-
[:INSTANCE_OF]->(g:goal{name:"user"}),
(dst:host{ip_addr="2.2.2.2"})<-[:AT_HOST]-(g2)-
[:INSTANCE_OF]->(g:goal{name:"user"})
WITH g1, g2 MATCH
```

```
p=(gi1)-[:NEXT*0..]->(middle:gi)-[:NEXT*0..]->(gi2)
WITH middle
MATCH (middle:gi)<-[:IMPLY]-()-[:REQUIRE]-()-[:EXPLOIT]->(v:vuln)
RETURN v
```

- Find those SAIs which are required in a maximum number of exploits. These nodes correspond to firewall rules and are possible candidate for blocking.

```
match (n:sai)-[r:REQUIRE]-()
with n
return n.Name, count(distinct r) as degree
order by degree desc
```

References

- Ammann, P., Wijesekera, D., and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, Washington, DC, November 18–22, 2002. pp. 217–224.
- Barik, M.S. and Mazumdar, C. 2014. A graph data model for attack graph generation and analysis. In *Proceedings of Second International Conference, SNDS 2014*, Trivandrum, India, March 13–14, 2014, Communications in Computer and Information Science Series, Volume 420, Springer-Verlag, Berlin, Heidelberg, pp. 239–250.
- Gremlin Query Language. 2016. Retrieved from <https://github.com/tinkerpop/gremlin/wiki>
- Ingols, K., Lippmann, R., and Piwowarski, K. 2006. Practical attack graph generation for network defense. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*. Washington, DC: IEEE Computer Society, pp. 121–130.
- Jajodia, S. and Noel, S. 2007. Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response. In *Algorithms, Architectures, and Information Systems Security*, B. Bhattacharya, S. Sur-Kolay, S. Nandy, and A. Bagchi (eds.). World Scientific Press, pp. 285–305.
- Jajodia, S., Noel, S., and O’Berry, B. 2003. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic (eds.), *Managing Cyber Threats: Issues, Approaches and Challenges*, Springer US, pp. 247–266.
- Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., and Cunningham, R. 2006. Validating and restoring defense in depth using attack graphs. In *Military Communications Conference, MILCOM 2006*, Piscataway, New Jersey: IEEE Press, pp. 981–990.
- Liu, C., Singhal, A., and Wijesekera, D. 2012. Using attack graphs in forensic examinations. In *Seventh International Conference on Availability, Reliability and Security (ARES)*, Prague, Czech Republic, pp. 596–603, 20–24.
- Neo4j. 2016. Retrieved from <http://www.neo4j.org/>
- Noel, S. and Jajodia, S. 2008. Optimal IDS sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management* 16(3): 259–275.
- Noel, S. and Jajodia, S. 2014. Metrics suite for network attack graph analytics. In R. K. Abercrombie, and J. T. McDonald (eds.), *Proceedings of the Ninth Annual Cyber and Information Security Research Conference*, New York: ACM, pp. 5–8.
- Noel, S., Harley, E., Him Tam, K., and Gyor, G. 2015. Big-data architecture for cyber attack graphs: Representing security relationships in NoSQL graph databases. In *IEEE Symposium on Technologies for Homeland Security (HST)*, Boston, Massachusetts, pp. 1–6.

- Noel, S., Jajodia, S., O'Berry, B., and Jacobs, M. 2003. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, pp. 86–95.
- Noel, S., Robertson, E., and Jajodia, S. 2004. Correlating intrusion events and building attack scenarios through attack graph distances. In *20th Annual Computer Security Applications Conference*, Tucson, Arizona, pp. 350–359.
- Ou, X., Boyer, W.F., and McQueen, M.A. 2006. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. New York: ACM, pp. 336–345.
- Phillips, C. and Swiler, L.P. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*. New York: ACM, pp. 71–79.
- Property Graph Model. 2016. Retrieved from <https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model>
- Ritchey, R.W. and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, pp. 156–165.
- Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, pp. 273–284.
- SPARQL Query Language for RDF. 2016. Retrieved from <https://www.w3.org/TR/rdf-sparql-query/>
- Swiler, L.P., Phillips, C., Ellis, D., and Chakerian, S. 2001. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II DISCEX '01*. Proceedings, Anaheim, California, pp. 307–321 vol.2.
- Swiler, L.P., Phillips, C., and Gaylor, T. 1998. *A Graph-Based Network-Vulnerability Analysis System*. Sandia National Laboratories, Albuquerque, New Mexico: ACM Press, pp. 97–110.
- The openCypher Project. 2016. Retrieved from <http://www.opencypher.org/>
- Wang, L., Noel, S., and Jajodia, S. 2006. Minimum-cost network hardening using attack graphs. *Journal of Computer Communications*, 29(18): 3812–3824.
- Wang, L., Yao, C., Singhal, A., and Jajodia, S. 2006. Interactive analysis of attack graphs using relational queries. In *Proceedings of the 20th IFIP WG 11.3 Working Conference on Data and Applications Security*. Berlin, Heidelberg: Springer-Verlag, pp. 119–132.
- Wang, L., Yao, C., Singhal, A., and Jajodia, S. 2008. Implementing interactive analysis of attack graphs using relational databases. *Journal of Computer Security*, 16(4): 419–437.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

16

Hands-On Aerospike

Khaleel Ahmad and Afsar Kamal

CONTENTS

16.1 Introduction.....	311
16.2 Features.....	312
16.3 Benefits.....	312
16.4 Installation Guide.....	313
16.5 Hardware Requirements.....	313
16.6 Aerospike Editions.....	313
16.7 Download and Setup.....	313
16.8 Install Aerospike Server (Community Edition 3.8.2.3).....	314
16.9 Create an Aerospike Virtual Machine.....	319
16.10 Run Aerospike.....	320
References.....	322

16.1 Introduction

Aerospike is the first flash-optimized, highly global-scale, and reliable in-memory NoSQL database for use. It is an open-source front-edge database that works with higher speed to make affordable for combining transactions and “hot” analytics to organize a large amount of information with strong ACID consistency and enterprise-grade reliability. A new class of Internet-scale, data-driven applications can be made easily and quick with the ability to process millions of transactions per second with sub-milliseconds of response time (<https://communities.intel.com/community/itpeernetwork/blog/2015/02/17/reaching-one-million-database-transactions-per-second-aerospike-intel-ssd>). Aim of Aerospike is growing with a new generation of interactive, real-time big data, Hadoop cluster, Web, and mobile applications. Aerospike’s fast path for high performance is key-value operations. And, extra features are incorporated such as strongly typed data model, secondary index queries, in-database computation with user-defined functions (UDFs), rich list and map interfaces, geographic replication, and a management console. Each line of code in Aerospike and every architectural decision are targeted on high performance and easily scaling operations, flexibility and scalability of database software, maximizing database performance, robustness, reliability, and robustness. All developers can take advantage of predictable, sub-millisecond latency for distributed caching, session management, user profile, and key-value store use cases. In nonstop production for nearly 4 years, Aerospike is deployed at real-time big data entrepreneurs and enterprises like AppNexus, eXelate, Chango, BlueKai, Inmobi, and Kayak and [x + 1] rely on Aerospike, so they focus on scaling their business without the complexity of building and maintaining a high-velocity database (<http://www.aerospike.com/open-source/>). Therefore, Aerospike

is the de facto choice for developers to build a new class of real-time and deliver instant, intuitive, and consistent experiences for multichannel marketing and telecommunication applications (https://www.micron.com/~media/documents/products/case-study/aerospike_ssd_case_study_lo.pdf).

Citrusleaf was the first name of Aerospike. The company renamed both the company and its software as Aerospike in August 2012 (<http://www.dbms2.com/2012/08/27/aerospike-the-former-citrusleaf/>). The name Aerospike is derived from Aerospike engine which is a type of rocket engine that maintains aerodynamics efficiency across a wide range of altitudes, expected software's scalability (http://www.hq.nasa.gov/office/pao/History/x-33/aero_faq.htm). In August 2012, Aerospike announced "NewSQL" database AlchemyDB that embraces the relational database model to solve the same scalability issues like NoSQL (<http://techcrunch.com/2012/08/28/nosql-company-citrusleaf-lands-series-b-funding-changes-name-and-acquires-alchemydb/>). On June 2014, the Aerospike database server is licensed under "Affero General Public License" (APGL) and Aerospike client software development kit (SDK) under the Apache License, Version 2.0 (http://www.theregister.co.uk/2014/06/24/aerospike_database_open_source/) (<http://siliconangle.com/blog/2014/06/24/aerospike-bets-on-end-users-to-monetize-subscriptions/?angle=silicon>) (<http://www.infoq.com/news/2014/06/aerospike-open-source>).

16.2 Features

Key-value store	It permits applications to maintain data devoid of a predefined schema. Records are indexed and stored with a key to quickly fetch the desired information from the database.
Flexible data model	It is capable to support voluminous and assortment of data being generated by the latest applications. Simple to combine data from multiple sources and not time consuming to update the existing data (http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling/).
UDFs	It is a piece of code written by developer to increase the database performance and functionality that runs inside the database server. Developers can easily create any complex data structure and function through the UDF.
Aggregation	Provides the ability to perform a chain of operations on a collection of records. Facilitates the flexibility to filter, transform, and aggregate the results of a query in distributed fashion using UDFs.
Query	Capability to perform value-based searches or to return a set of records through the use of fast, high-concurrency queries on secondary indexes (http://www.infoworld.com/article/2609739/database/scaling-and-querying-large-nosql-databases.html).

16.3 Benefits

Open source	Free for developers to use Aerospike and build their own database, not just the few who works at company.
Flash optimized	To achieve better performance and superior speed with less operation and lower infrastructure costs being a hybrid RAM/SSD storage architecture.
Scalable	It is the ability to add data, compute, and throughput easily—without downtime or degrading performance.
High availability	Durability to operate continuously 100% without failure for a long time or never failing.

Continued

High-performance NoSQL	Provides tens of millions of accesses per second or billions of accesses per second and millions of inserting/ updating/ deleting per second.
Easy to get started	Depends on developer to choose desired programming language as a client and install Aerospike Server on any OS environment.
World-class support	24 × 7 support by senior developers and operations experts to monitor applications and maintain top performance of clusters for customer satisfaction from anywhere in the world.

16.4 Installation Guide

Basically, Aerospike is designed for 64-bit Linux, but it accepts suitable Linux distribution as .rpm packages for Red Hat variants and .deb packages for Ubuntu and Debian. Vagrant-managed virtual machines support OS X and Windows. Through the Vagrant cloud virtual machine, it is easy to download and run the Aerospike using few simple commands.

16.5 Hardware Requirements

1. CPU: Supporting Linux or Linux virtual machine can be used, or install on 4 core and 8 core Intel Xeon hardware as well as on AMD, but recommended is Single 4 core processor.
 2. Memory: Minimum 4G of DRAM is required.
 3. Storage: 8 times the amount of data stored in RAM.
 4. Network: Any TCP network as 1G, 10G, or better. Multicast IP can be used for ease of installation.
-

16.6 Aerospike Editions

There are two types of database packages of Aerospike:

1. Community Edition (free edition)
 2. Enterprise Edition
-

16.7 Download and Setup

Here, we describe all the steps, processes, or procedure based on Windows OS:

1. Aerospike is supported on virtual machine managed by Vagrant. So, you need to install Vagrant.

- If you have already installed the Vagrant, then run following commands to restart Aerospike.

```
vagrant up
vagrant ssh
sudo service aerospike start
sudo service amc start
```

and to stop asd and halt the virtual machine, run this command:

```
vagrant halt
```

- Otherwise, download Vagrant through this link: https://releases.hashicorp.com/vagrant/1.8.1/vagrant_1.8.1.msi
- Vagrant cannot work without a virtualization system such as Oracle virtualBox or VMWare, so you need to install VirtualBox.
- Install VirtualBox through this link: http://download.virtualbox.org/virtualbox/5.0.20/Oracle_VM_VirtualBox_Extension_Pack-5.0.20-106931.vbox-extpack

16.8 Install Aerospike Server (Community Edition 3.8.2.3)

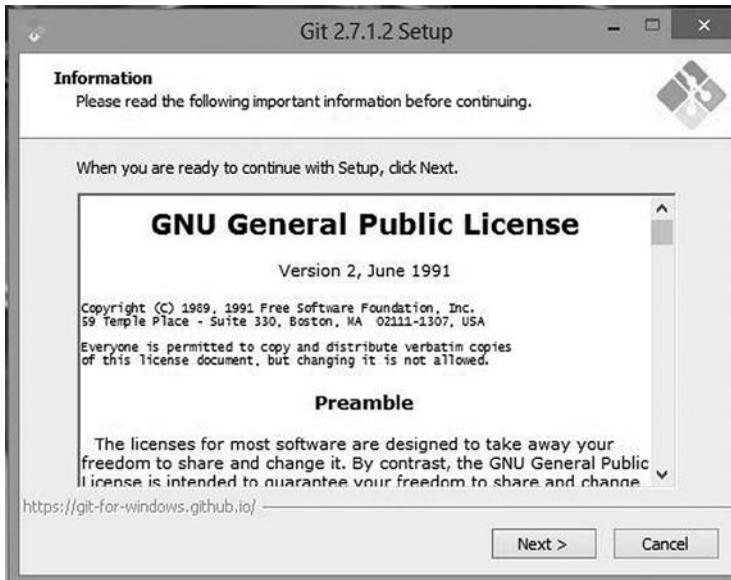
Packages for different Linux distribution are available in Aerospike Server Community Edition with a tools installer and a server installer. These are as follows:

Packages	Links to download
Redhat 6 (.rpm packages)	http://www.aerospike.com/download/server/3.8.2.3/artifact/el6
Redhat 7 (.rpm package)	http://www.aerospike.com/download/server/3.8.2.3/artifact/el7
Ubuntu 12.04+ .deb packages	http://www.aerospike.com/download/server/3.8.2.3/artifact/ubuntu12
Ubuntu 14.04+ .deb packages	http://www.aerospike.com/download/server/3.8.2.3/artifact/ubuntu14
Debian 7 .deb packages	http://www.aerospike.com/download/server/3.8.2.3/artifact/debian7
Debian 8 .deb packages	http://www.aerospike.com/download/server/3.8.2.3/artifact/debian8
Linux .tar.gz package	http://www.aerospike.com/download/server/3.8.2.3/artifact/tgz
Mac OS X Vagrant Box	http://www.aerospike.com/docs/operations/install/vagrant/mac/
Windows Vagrant Box	http://www.aerospike.com/docs/operations/install/vagrant/win/
Amazon EC2 Cloud	http://www.aerospike.com/docs/deploy_guides/aws/install
Google Compute Engine Cloud	http://www.aerospike.com/docs/deploy_guides/gce/install

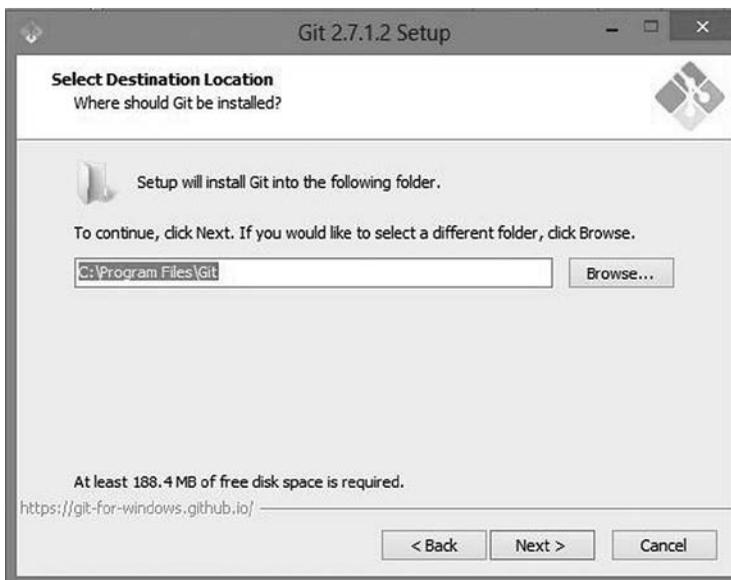
To install on Windows, follow step-by-step process:

- Download GIT through this link: <https://github.com/git-for-windows/git/releases/download/v2.8.2.windows.1/Git-2.8.2-64-bit.exe>

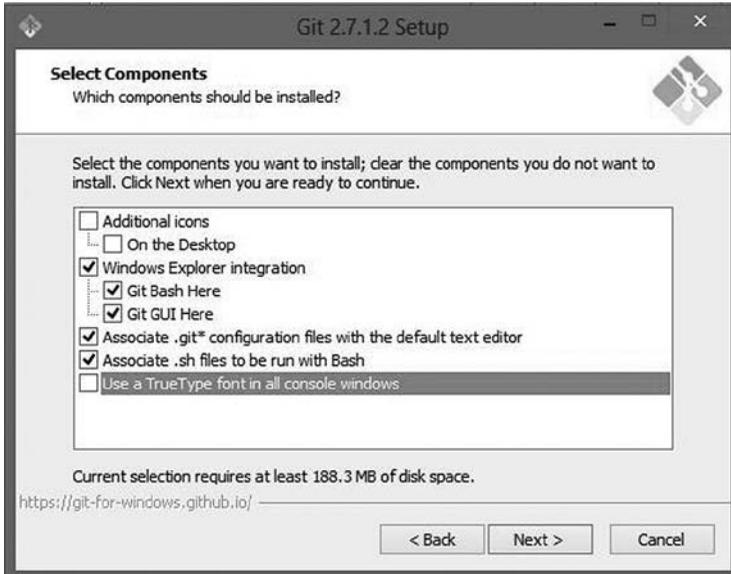
2. Run the application after completion of downloading, a setup will appear, read the terms and condition carefully, and then click on the Next button.



3. Select your destination folder clicking on the browse button where you want to install the Git and click on the Next button.



4. Select or unselect the component as per your requirements for installation, and then click on the Next button.



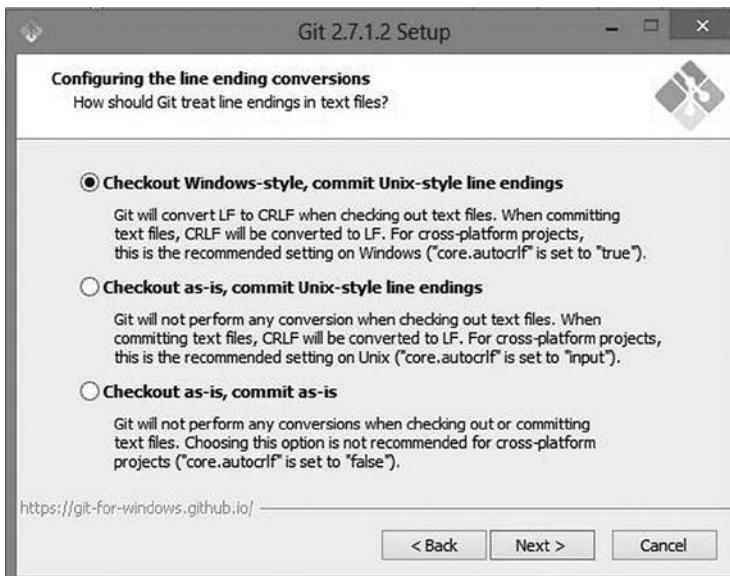
5. Choose the Start Menu folder to place the program shortcut or check the box to stop the creation of Start Menu folder and click on *Next* button.



6. Check the radio button based on the needs to adjust the path environment, and then click on the *Next* button.



7. Set the configuration for line-ending conversions. Select the Radio button for choosing your own style.



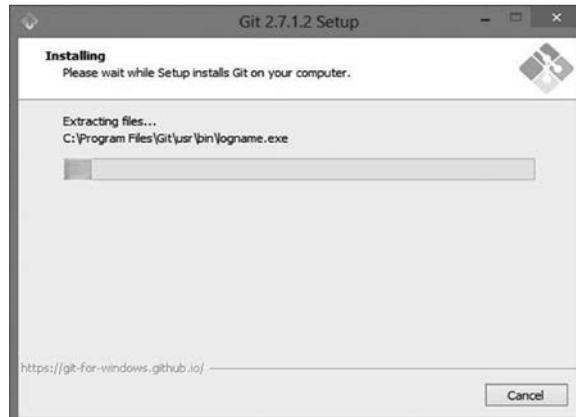
8. Select a suitable Radio button to set the terminal to work with *Git Bash* after reading the instruction.



9. If you want to read the data in bulk and being operations in cached memory, then check the box and click on the Next button.



- Installation will be processed. So, wait until the installation is completed.



- Click on *Finish* button to finish the setup.



The next steps will request commands be entered into the command prompt, which simply requires you to run Git Bash from the directory specified.

16.9 Create an Aerospike Virtual Machine

- Create an Aerospike Directory:
Create a working directory anywhere for each virtual machine through this command:

```
mkdir ~/aerospike-vm && cd ~/aerospike-vm
```

```
kamal@DESKTOP-UF4UGCS: ~$ cd ~
kamal@DESKTOP-UF4UGCS: ~$ mkdir ~/aerospike-vm && cd ~/aerospike-vm
mkdir: cannot create directory '/c:/Users/kamal/aerospike-vm': File exists
```

(Directory not created, I am already working with Aerospike in this directory.)
All Vagrant commands will run within this directory to manage the virtual machine.

2. To initialize the Aerospike Virtual machine, run this command:

```
vagrant init aerospike/centos-6.5
```

```
kamal@DESKTOP-UF4UGCS MINGW64 ~
$ vagrant init aerospike/centos-6.5
Vagrantfile already exists in this directory. Remove it before
running 'vagrant init'
```

(Vagrant file is also running.)

16.10 Run Aerospike

1. To start up the virtual machine and asd, run this command.

```
vagrant up
```

```
kamal@DESKTOP-UF4UGCS MINGW64 ~
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'aerospike/centos-6.5'...
Progress: 40%
```

2. Verify whether the asd and AMC have started successfully or not, and run this command:

```
vagrant ssh -c "sudo service aerospike status"
```

This will give you the message whether asd is running or connection is closed.

```
vagrant ssh -c "sudo service amc status"
```

Message will be displayed as AMC is running, connection is closed, or retrieving AMC status.

You can also search the server log, and run this command:

```
vagrant ssh -c "sudo grep -i cake /var/log/aerospike/aerospike.log"
```

Displays a message either port is ready or not with date and time zone.

3. Verify that the Aerospike is successfully installed on your system and is running.
 - a. The essential files are placed in these directory after installing the Aerospike.

<i>/etc/aerospike/</i>	<i>- configuration files for Aerospike</i>
<i>/etc/aerospike/aerospike.conf</i>	<i>- default configuration for Aerospike</i>
<i>/etc/init.d/aerospike</i>	<i>- init script for Aerospike</i>

```

/etc/logrotate.d/aerospike    - logrotate configuration for
                              Aerospike
/opt/aerospike/bin/          - binaries including Aerospike
                              server and tools
/opt/aerospike/doc/          - documents, including licenses
/opt/aerospike/sys/          - system data files, maintained by
                              Aerospike
/opt/aerospike/usr/          - user data files
/var/log/aerospike/          - log files emitted by Aerospike
/usr/bin/asd                  - Aerospike Server daemon

```

b. Verify record operations

As creating a new object adding two fields “book” and “publisher” with a key “Aerospike” in the “test” that is a part of default configuration.

```
cli -h 127.0.0.1 -n test -o set -k Aerospike -b book -v "Calculus,
Math."
```

succeeded: key= Aerospike set= bin= book value= Calculus, Math.

```
cli -h 127.0.0.1 -n test -o set -k Aerospike -b publisher -v
"Vyasa Publication, Hyd"
```

succeeded: key= Aerospike set= bin= publisher value= Vyasa Publication, Hyd

Delete the record and verify:

```
$ cli -h 127.0.0.1 -n test -o delete -k Aerospike
```

delete succeeded: key= Aerospike set=

c. Develop the Application after downloading an acceptable language from Aerospike Client library that includes source or binary for the language, examples and Benchmark tools.

S. No.	Client Library
1	C / C++ 4.0.4 C
2	Java 3.2.2 Java
3	Go 1.13.0 Go
4	C# / .NET 3.2.2 C#
5	Node.js 2.0.4 Node.js
6	C libevent 2.1.44 C

Continued

S. No.	Client Library
7	PHP 3.4.8 PHP
8	Erlang 2.1.2 Erlang
9	Python 2.0.3 Python
10	Ruby 1.0.12 Ruby
11	Perl 2.1.34 Perl
12	JDBC 0.2.3 Aerospike JDBC Connector

Each client library provides various certified packages on different platform. Download right and suitable file based on programming language for application.

References

- Aero_FAQ (n.d.). Retrieved May 12, 2016, from http://www.hq.nasa.gov/office/pao/History/x-33/aero_faq.htm
- Aerospike Open Sources NoSQL Database in Bid to Expand Customer Ranks (n.d.). Retrieved April 25, 2016, from <http://siliconangle.com/blog/2014/06/24/aerospike-bets-on-end-users-to-monetize-subscriptions/?angle=silicon>
- Aerospike Open Sources Their In-Memory NoSQL Database (n.d.). Retrieved May 1, 2016, from <http://www.infoq.com/news/2014/06/aerospike-open-source>
- Aerospike: Thanks for that \$20m, VCs ... next we'll OPEN SOURCE our NoSQL database (n.d.). Retrieved May 12, 2016, from http://www.theregister.co.uk/2014/06/24/aerospike_database_open_source/
- Aerospike, the Former Citrusleaf (n.d.). Retrieved May 28, 2016, from <http://www.dbms2.com/2012/08/27/aerospike-the-former-citrusleaf/>
- Finley, K. 2012. Grim and Gritty Startup Reboot: NoSQL Company Citrusleaf Changes Name and Acquires AlchemyDB. Retrieved May 12, 2016, from <http://techcrunch.com/2012/08/28/nosql-company-citrusleaf-lands-series-b-funding-changes-name-and-acquires-alchemydb/>
- NoSQL Applications Take Flight with Aerospike Flash ... (n.d.). Retrieved June 12, 2016, from https://www.micron.com/~media/documents/products/case-study/aerospike_ssd_case_study_lo.pdf
- NoSQL Data Modeling. 2014. Retrieved April 27, 2016, from <http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling/>
- Open Source - | Aerospike. (n.d.). Retrieved June 1, 2016, from <http://www.aerospike.com/open-source/>
- Reaching one million database transactions per second ... Aerospike Intel SSD - IT Peer Network. 2015. Retrieved June 11, 2016, from <https://communities.intel.com/community/itpeernetwork/blog/2015/02/17/reaching-one-million-database-transactions-per-second-aerospike-intel-ssd>
- Scaling and Querying Large NoSQL Databases. 2013. Retrieved May 20, 2016, from <http://www.infoworld.com/article/2609739/database/scaling-and-querying-large-nosql-databases.html>

17

Hands-On Cassandra for Windows

Khaleel Ahmad and Abdul Haseeb

CONTENTS

17.1 Introduction.....	323
17.2 Tools and Utilities Provided by Cassandra.....	329
17.3 CQL Shell	330
17.4 NodeTool	330
References.....	332

17.1 Introduction

Cassandra is one of the widely scalable open-source NoSQL databases. Cassandra is best for handling gigantic amounts of structured, semistructured, and unstructured data across distributed data centers and the cloud. Cassandra can provide continuous availability, linear scalability, and operational simplicity across many commodity servers and not using a single point of failure, together with a robust dynamic data model designed for the highest flexibility and quick-response instances [1,2].

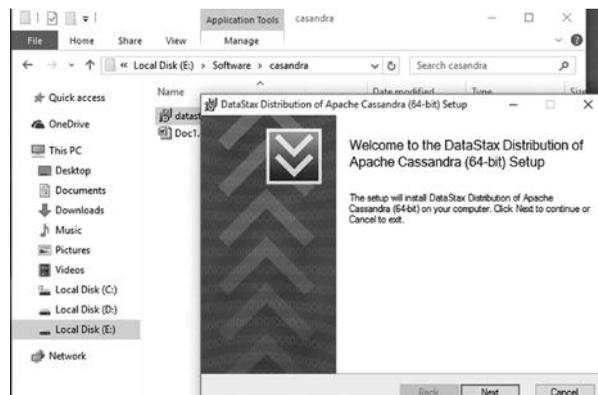
Links of NoSQL Cassandra

<http://www.planetcassandra.org/cassandra/> (for offline)

<https://academy.datastax.com/tutorials> (for offline)

<http://www.planetcassandra.org/try-cassandra/> (for online)

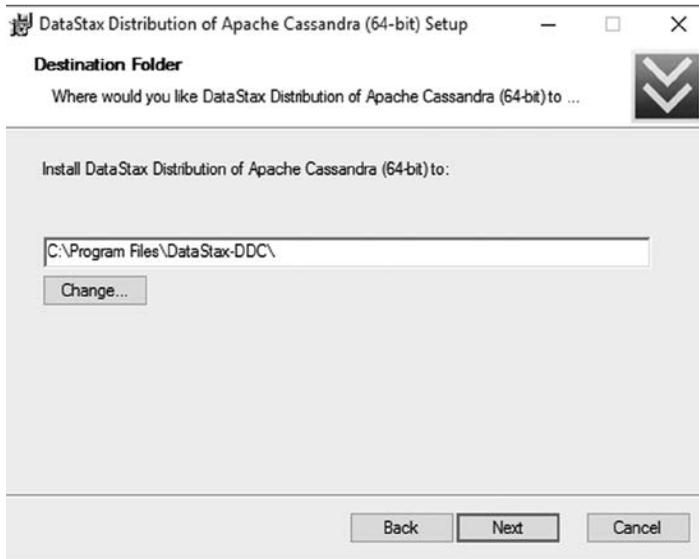
Step 1: First download the appropriate MSI from Planet Cassandra from the above given links. Run the software, and run it to launch the setup wizard which looks like.



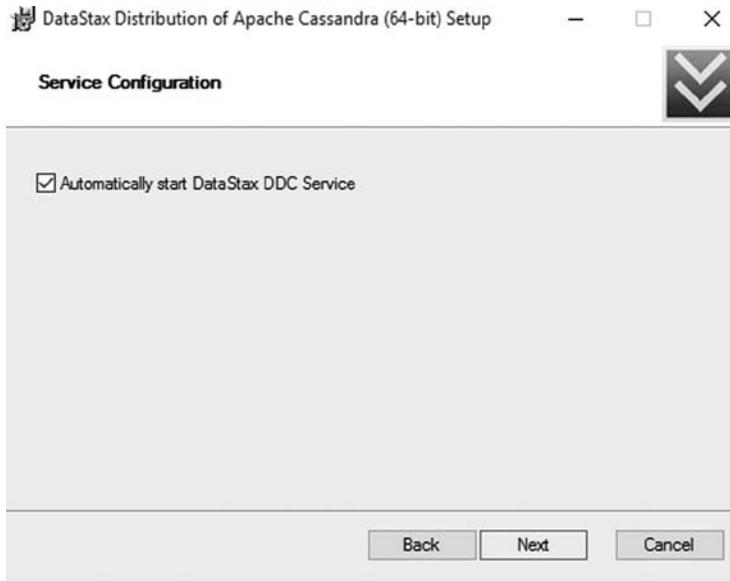
Step 2: End User License Agreement screen will appear, select End User License Agreement checkbox, and then click on the next button.



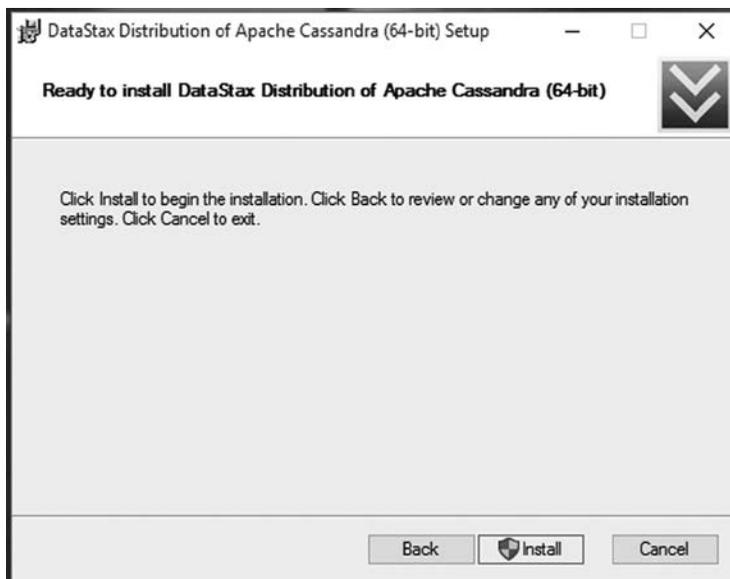
Step 3: Select your destination folder by clicking on the Change button where you want to install the Cassandra, and click on the Next button.



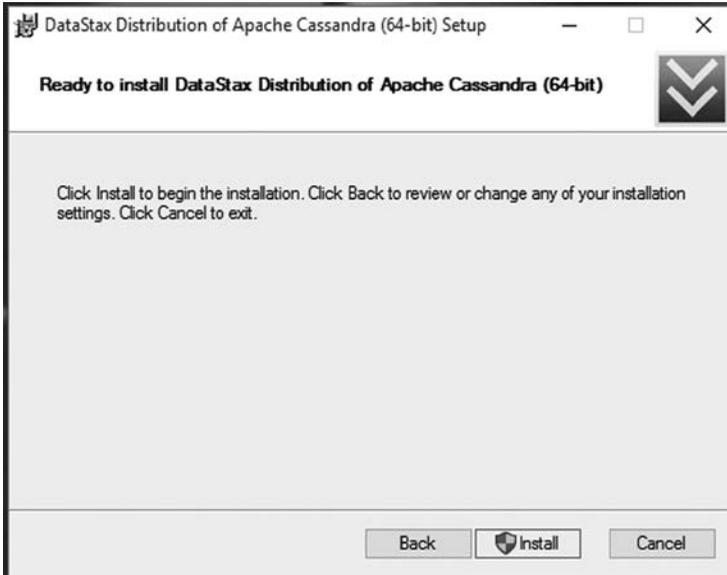
Step 4: Here, select checkbox for DataStax Distribution of Apache Cassandra (DDC) that configures the services automatically. Now, click the Next button.



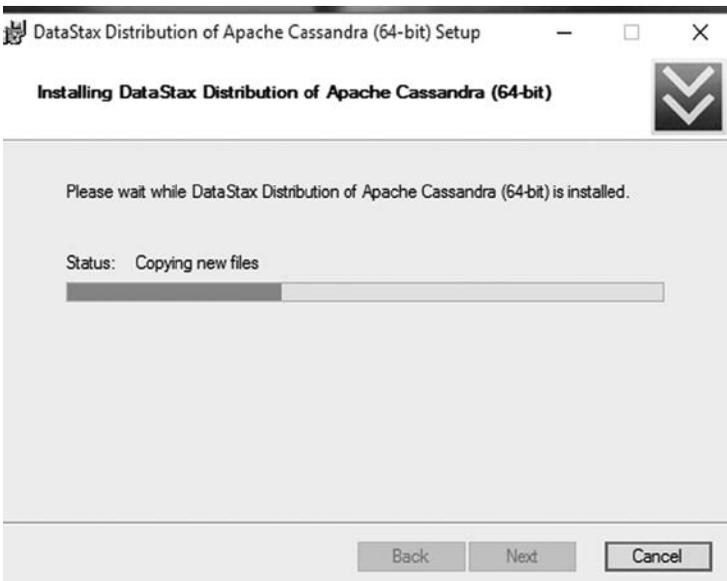
Step 5: Here, ready to install DataStax Distribution of Apache Cassandra (64-bit), and you just click the Install button.



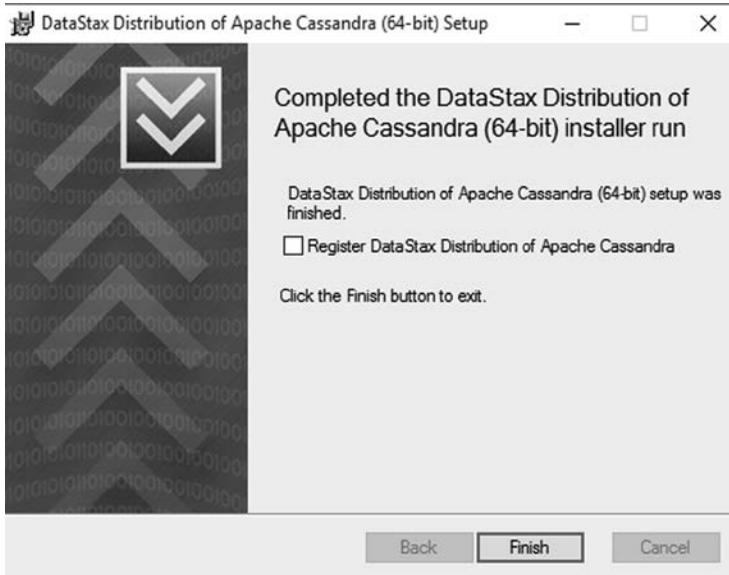
Step 6: If you want to change the installation setting, then you click on the Back button; if not so, then click Install button.



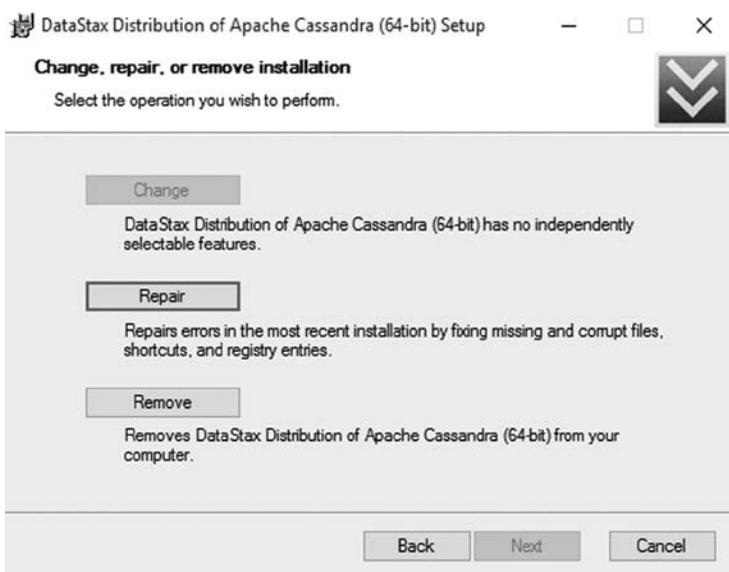
Step 7: If you want to change the installation setting, then you just click on the Back button; if not so, then click on Next button.



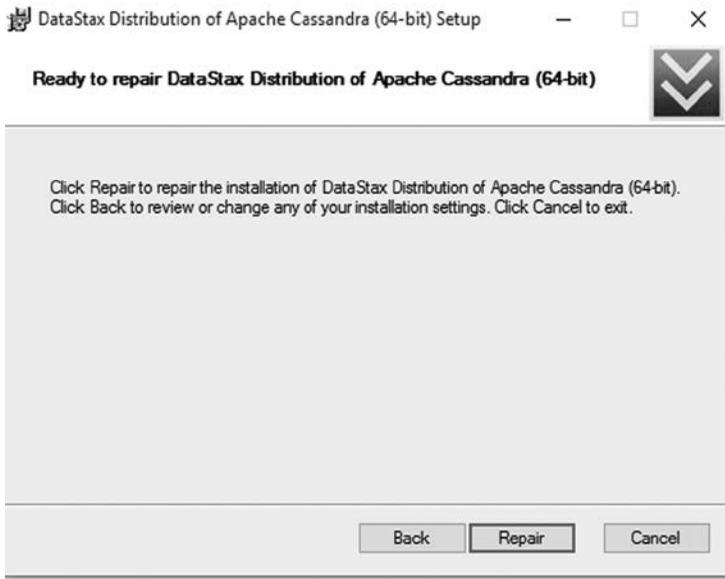
Step 8: After completing the installation process, click on *Finish* button to finish the setup.



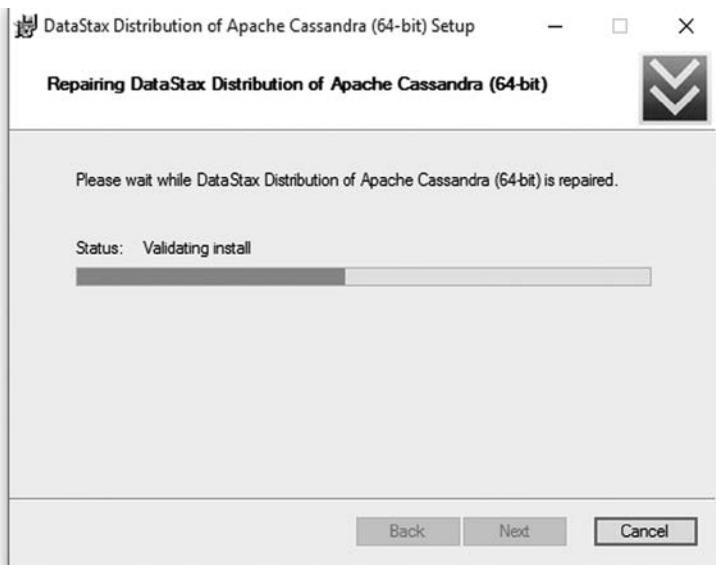
Step 9: *Repair Software*: Click on the setup files of Cassandra for Repair/Change/Remove. Select the *Repair* button, and click on *Next* button.



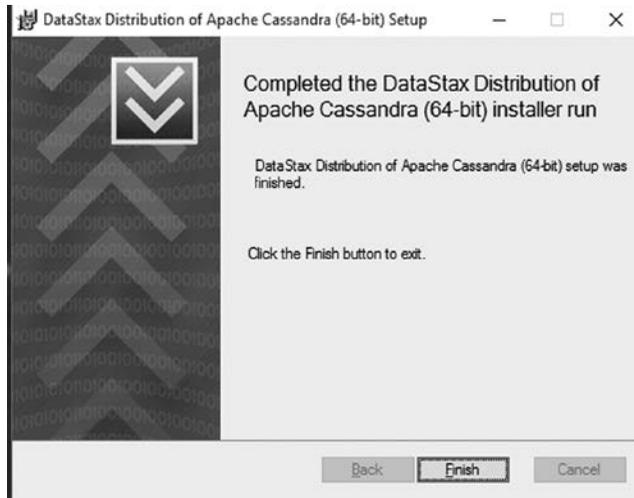
Step 10: After getting the screen of “Ready to repair DataStax Distribution of Apache Cassandra”, click on *Repair* button.



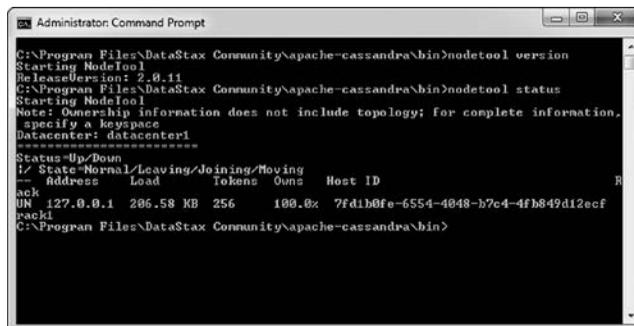
Step 11: Apache Cassandra is repairing on. So, wait until the repairing process is completed. After completing repair process, click on *Next* button.



Step 12: Finally, click on finish button, and the software is completely repaired.



Step 13:



There are three services installed:

- *DataStax Cassandra Community Server*: It is the Cassandra database itself.
- *DataStaxOpsCenter Agent*: Agent at OpsCenter fetches health or statistics information from cluster.
- *DataStaxOpsCenter Community*: OpsCenter program collects information from the agents and gives the web UI to view the health information and manage your cluster.

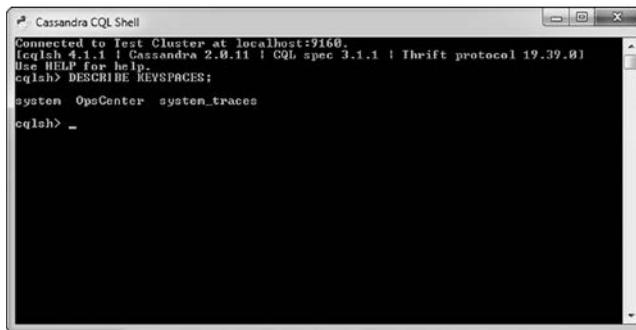
17.2 Tools and Utilities Provided by Cassandra

All of the tools and utilities are installed on Windows by default under the below folder.
 C:\Program Files\DataStax Community\apache-cassandra\bin

17.3 CQL Shell

CQL Shell stands for Cassandra query language, or as more commonly known as `cqlsh`, this is a REPL for running the commands on CQL statements against a Cassandra cluster interactively. Now, when we install Cassandra on Windows with the installer setup file, we will get automatically the Start Menu link (under the DataStax Community Edition folder) to launch it. We can also open it from the command line using `cqlsh.bat` from the aforementioned Cassandra bin directory. Now if launching from the command line, we can use the command `-h` or `--help` flag to see all of the functions available on the popup window.

By default, `cqlsh` will connect to the Cassandra node on local host. We can use CQL Shell to check if newly installed Cassandra cluster is running properly or not. For example, we could run the `DESCRIBE KEYSPACES` command to list the Keyspaces currently available in our cluster (Introduction to Cassandra Query Language, n.d.).



```

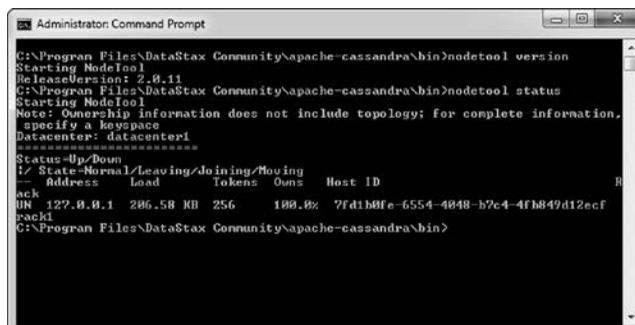
Cassandra CQL Shell
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.11 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh> DESCRIBE KEYSPACES;
system OpsCenter system_traces
cqlsh> _

```

17.4 NodeTool

NodeTool is the Swiss army knife of tools for Cassandra. A ton of commands can be acquired for managing the clusters. We can search it beneath the aforementioned Cassandra bin directory.

NodeTool is a command line tool. We cannot search a Start Menu link for beginning it. Endeavor running `nodetool.bat`, and see the list of commands that are available.



```

Administrator: Command Prompt
C:\Program Files\DataStax Community\apache-cassandra\bin>nodetool version
Starting NodeTool
ReleaseVersion: 2.0.11
C:\Program Files\DataStax Community\apache-cassandra\bin>nodetool status
Starting NodeTool
Note: Ownership information does not include topology; for complete information,
specify a keypace
Datacenter: datacenter1
*****
Status=Up/Down
|-/ State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID
ack 127.0.0.1 206.58 KB 256 100.0% 7fd1b9fe-6554-4048-b7c4-4fb849d12ecf
pack1
C:\Program Files\DataStax Community\apache-cassandra\bin>

```

Command Name-CREATE KEYSPACE Tutorialpoint

```
WITH replication = {'class':'SimpleStrategy', 'replication_factor': 3};
```

Command Name-DESCRIBE KEYSACES;**Command Name-Use Tutorialpoint system system_traces ...;****Command Name-CREATE TABLE emp (**

```
... firstname text,
... lastname text,
... ageint,
... email text,
... city text,
... PRIMARY KEY (lastname));
```

Command Name-Select * from emp;

```

Cassandra CLI Shell
Connected to Test Cluster at 127.0.0.1:9042.
(cqlsh 3.0.1 | Cassandra 3.0.0 | CQL spec 3.4.0 | native protocol v4)
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh> CREATE KEYSPACE 'keySpace Name';
... WITH replication = {'class':'SimpleStrategy', 'replication_factor': 3};
cqlsh> DESCRIBE keyspaces;
Tutorialpoint system_auth system_distributed
system_schema system system_traces
cqlsh> use Tutorialpoint;
cqlsh> CREATE TABLE emp;
... emp_id int PRIMARY KEY,
... emp_name text,
... emp_city text,
... emp_sal varint,
... emp_phone varint
cqlsh> select * from emp;
emp_id | emp_city | emp_name | emp_phone | emp_sal
-----|-----|-----|-----|-----
(0 rows)
cqlsh> INSERT INTO emp VALUES(1, 'Hyderabad', 'Haseeb', 7416770382, 50000);
SyntaxException: (ErrorMessage code=2000 [Syntax error in CQL query] message="line 1:16 no viable alternative at input 'values' (insert into [emp] values...)"
cqlsh> INSERT INTO emp VALUES(1, 'Hyderabad', 'Haseeb', 7416770382, 50000);
SyntaxException: (ErrorMessage code=2000 [Syntax error in CQL query] message="line 1:16 no viable alternative at input 'values' (insert into [emp] values...)"

```

**CommandName-INSERTINTOemp(emp_id,emp_city,emp_name,emp_phone,emp_sal)
VALUES (1, 'Hyderabad', Haseeb', 7416770382,'50000')
USING TIMESTAMP 123456789;**

```

Cassandra CLI Shell
SyntaxException: (ErrorMessage code=2000 [Syntax error in CQL query] message="line 1:16 no viable alternative at input 'insert' ([insert...])"
cqlsh> INSERT INTO emp USING TIMESTAMP 123456789;
SyntaxException: (ErrorMessage code=2000 [Syntax error in CQL query] message="line 1:16 no viable alternative at input 'insert' ([insert...])"
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES(1, 'Hyderabad', 'Haseeb', 7416770382, 50000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES (1, 'Hyderabad', 'Haseeb', 7416770382, 50000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES (1, 'Hyderabad', 'Haseeb', 7416770382, 50000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES (1, 'Hyderabad', 'Haseeb', 7416770382, 50000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> select * from emp;
emp_id | emp_city | emp_name | emp_phone | emp_sal
-----|-----|-----|-----|-----
1 | Hyderabad | Haseeb | 7416770382 | 50000
(1 rows)
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES (2, 'Delhi', 'Javed', 9528895404, 10000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> INSERT INTO emp(emp_id,emp_city,emp_name,emp_phone,emp_sal) VALUES (3, 'Mumbai', 'Shadab', 9308022091, 20000) USING TTL 86400;
InsertRequest: (ErrorMessage code=2000 [Invalid query] message="keyspace Tutorialpoint does not exist")
cqlsh> select * from emp;
emp_id | emp_city | emp_name | emp_phone | emp_sal
-----|-----|-----|-----|-----
1 | Hyderabad | Haseeb | 7416770382 | 50000
2 | Delhi | Javed | 9528895404 | 10000
3 | Mumbai | Shadab | 9308022091 | 20000
(3 rows)
cqlsh> UPDATE emp SET user_name = 'bob';
SyntaxException: (ErrorMessage code=2000 [Syntax error in CQL query] message="line 1:16 mismatched input ';' expecting <WHERE>")
cqlsh> select * from emp;
emp_id | emp_city | emp_name | emp_phone | emp_sal
-----|-----|-----|-----|-----
1 | Hyderabad | Haseeb | 7416770382 | 50000
2 | Delhi | Javed | 9528895404 | 10000
3 | Mumbai | Shadab | 9308022091 | 20000
(3 rows)
cqlsh>

```

References

1. About Apache Cassandra (n.d.). Retrieved June 25, 2016, from https://docs.datastax.com/en/cassandra_win/2.2/cassandra/cassandraAbout.html
2. Introduction to Cassandra Query Language (n.d.). Retrieved June 15, 2016, from http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html

18

Hands-On Cloudant

Masroor Ansari and Khaleel Ahmad

CONTENTS

18.1 Introduction.....	333
18.2 IBM Cloudant NoSQL Database Capabilities.....	333
References.....	340

18.1 Introduction

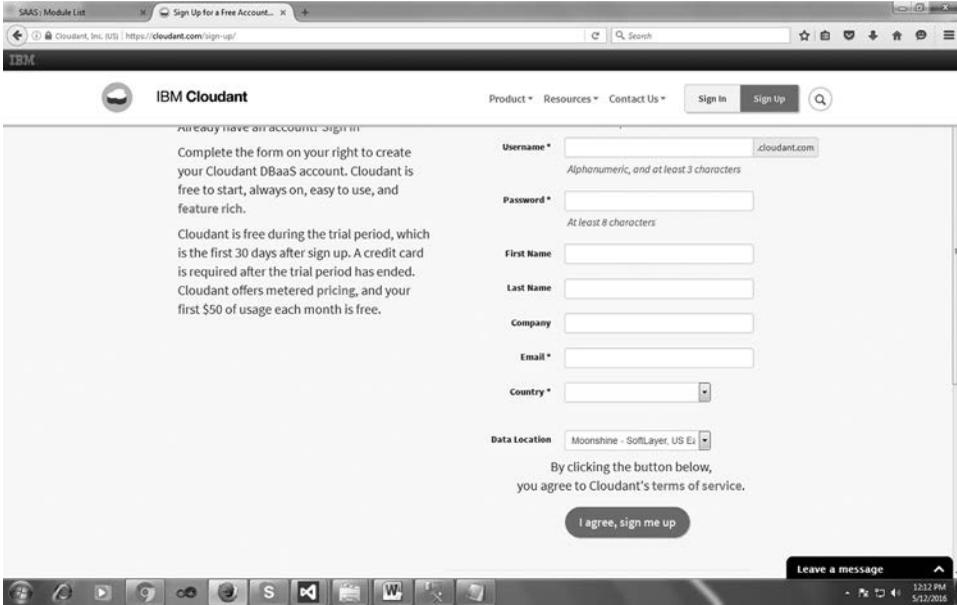
IBM Cloudant is a NoSQL database platform dedicated to the cloud. You may use Cloudant as a completely controlled database as a server (DBaaS) running on public cloud platforms like IBM SoftLayer or through an on-premise edition known as Cloudant Local. You can deploy Cloudant in one of the modes like private, public, or hybrid cloud [1,2].

18.2 IBM Cloudant NoSQL Database Capabilities

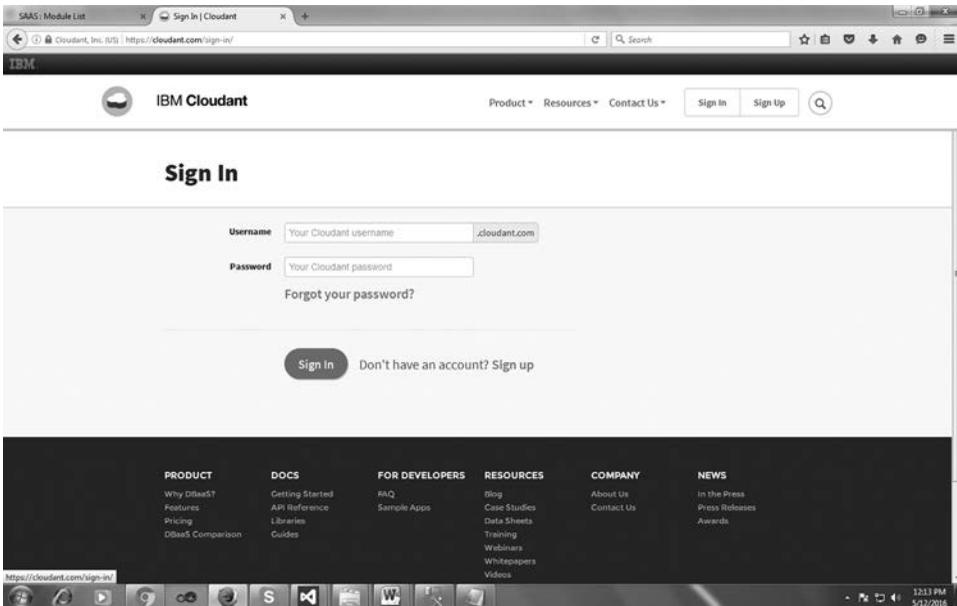
- *JSON data store*: JavaScript Object Notation (JSON) stores data as self-describing documents and makes it easy to handle multistructured data and modify schemas.
- *Incremental MapReduce*: Incremental MapReduce generates secondary indexes on JSON data which might be up to date in real time and execute advanced aggregate functions on data.
- *Geospatial function*: This efficaciously influences analysis of geographic information system data to enhance apps with vicinity services viz. predictive routes, route optimization, and bounding polygons.
- *Search*: Dominant Apache Lucene-based full-text search together with phrase, proximity, wild-carded, and fuzzy queries.
- *Management and monitoring*: Monitoring system and management infrastructure monitor supporting quicker response to environmental changes.
- *Security*: It provides the security-rich needs for the applications [2].

Follows *step-by-step process* to run IBM Cloudant database. How to create database, documents, queries, views, indexes, etc.

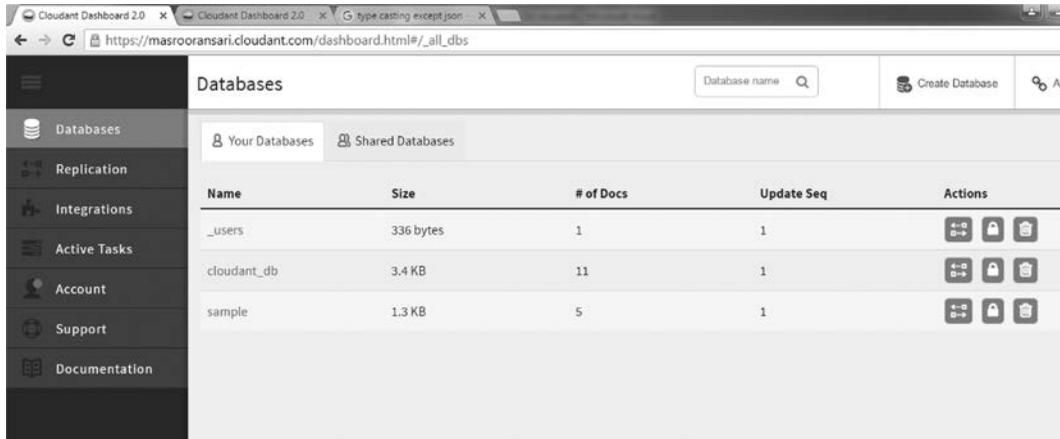
Step 1: First *sign-up* for Cloudbant database by using the following link:
<https://cloudant.com/sign-up>.



Step 2: After creating an account, *login* to your account.



Step 3: There are a number of TABS available on Dashboard page viz. databases, replication, warehousing, active tasks, account, and support.



Step 4: Creation of document:

_id: Design Document ID

_rev: Design Document Revision

Views: An object represents MapReduce views

Index name: Index definition (one for each index)

Indexes: An object describing search indexes

Stopwords (optional): The default list of stop words for the analyzer is given below:

"a", "an", "as", "at", "are", "and", "be", "but", "for", "in", "is", "it", "if", "into", "no", "not", "on", "of", "or", "such", "to", "the", "this", "that", "there", "their", "these", "they", "then", "with", "will", "was" (https://docs.cloudant.com/design_documents.html).

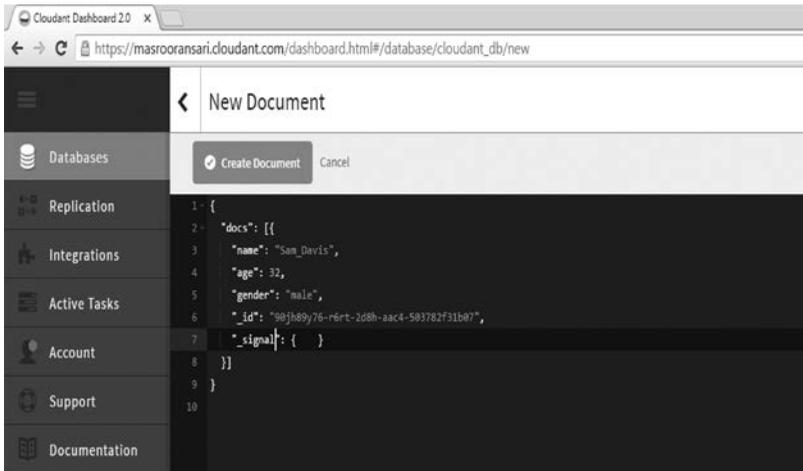
Examples: There are two query examples for document creation as given below (<https://developer.ibm.com/clouddataservices/docs/cloudant/cloudant-index-types-defined/use-cloudant-query/>).

Example1:

```
{
  "prices": [ {
    "Reliance Mega Mall": 6.12,
    "Price Max": 67.23,
    "_Express": 3.55,
    "_id": "453m34-34ne-43nw-34dedfef",
    "_item": "Britannia"
  } ]
}
```

Example2:

```
{
  "docs": [ {
    "name": "Sam_Davis",
    "age": 32,
    "gender": "male",
    "_id": "90jh89y76-r6rt-2d8h-aac4-503782f31b07",
    "_signal": { }
  } ]
}
```



Step 5: Creation of query indexes:

Cloudant Query is JSON query syntax for Cloudant databases. Cloudant Query envelopes numerous index types starting with the important index out of the box. Cloudant Query indexes can be constructed by MapReduce Views, where the index type is JSON, and Search Indexes, where the index type is text.

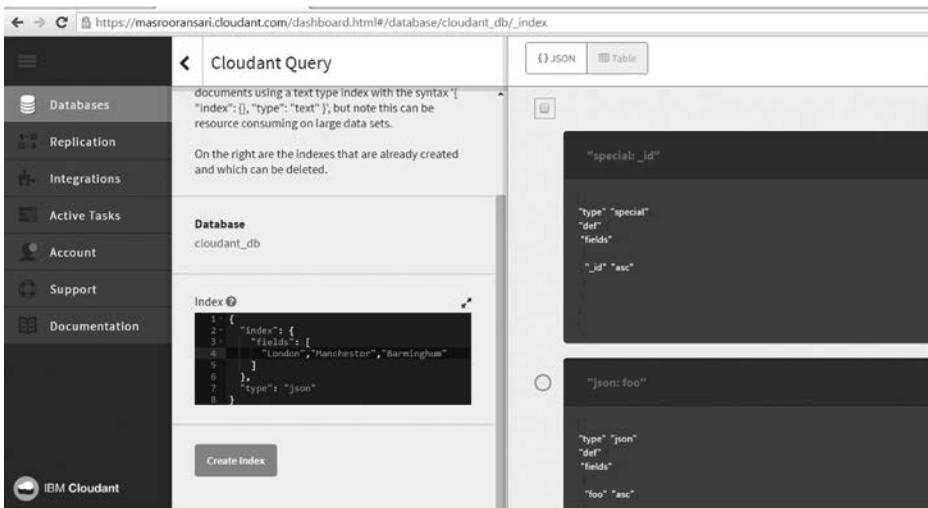
There are two query examples for document creation as given below.

Example1:

```
{
  "index": {
    "fields": [
      "India", "China", "Japan", "Hongkong"
    ]
  },
  "type": "json"
}
```

Example2:

```
{
  "index": {
    "fields": [
      "London", "Manchester", "Barmingham"
    ]
  },
  "type": "json"
}
```



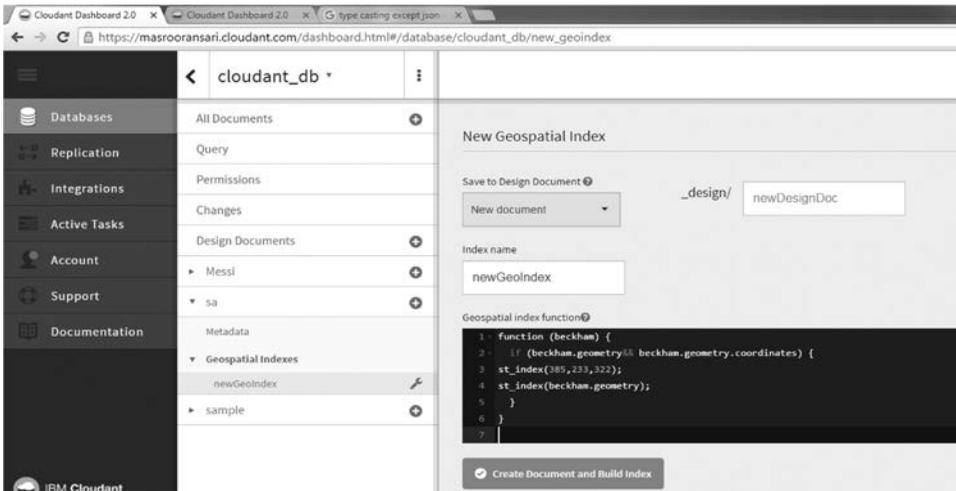
Step 6: Creation of geospatial indexes

Example 1:

```
function (messi) {
  if (messi.geometry && messi.
    geometry.coordinates) {
    st_index(messi.geometry);
  }
}
```

Example 2:

```
function (beckham) {
  if (beckham.geometry &&
    beckham.geometry.coordinates) {
    st_index(385,233,322);
    st_index(beckham.geometry);
  }
}
```



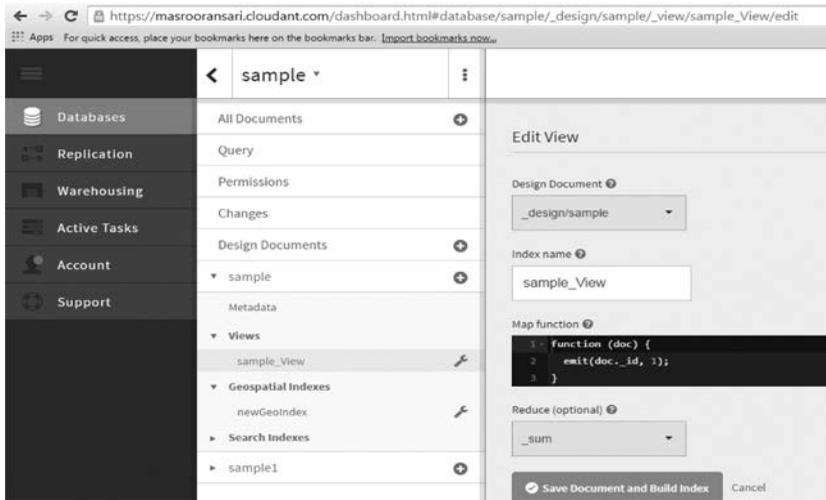
Step 7: Creation of view

Example 1:

```
function (beckham) {
  emit(beckham._id, 1);
}
```

Example 2:

```
function (keys, reproduce) {
  if (reproduce) {
    i=56;
    j=98.09;
    values=j+k;
    return sum(values);
  } else {
    return values.length;
  }
}
```



Step 8: Copy from one database to another database or delete or replicate

The structure of the copy command

Method: COPY /cloudant_db/_design/newGeoIndex-doc

Request: None

Response: JSON represents revision and new document

Roles permitted: _writer

Query Arguments:

Argument: rev

Description: Revision to copy from

Optional: yes

Type: string

HTTP Headers

Header: Destination

Description: Destination document (and optional revision)

Optional: no

The structure of the delete command

Method: DELETE /cloudant_db/_design/newGeoIndex-doc

Request: None

Response: JSON of deleted design document

Roles permitted: _writer

Query Arguments:

Argument: rev

Description: Current revision of the document for validation

Optional: yes

Type: string

HTTP Headers**Header:** If-Match**Description:** Current revision of the document for validation**Optional:** yes**Example 1:**

```

COPY / cloudant_db
/_design/sample?rev=1-
e23b9e942c19e9fb10ff1fde2e50e0f5
HTTP/1.1
Content-Type: application/JSON
Destination: recipes/_design/
  recipelist

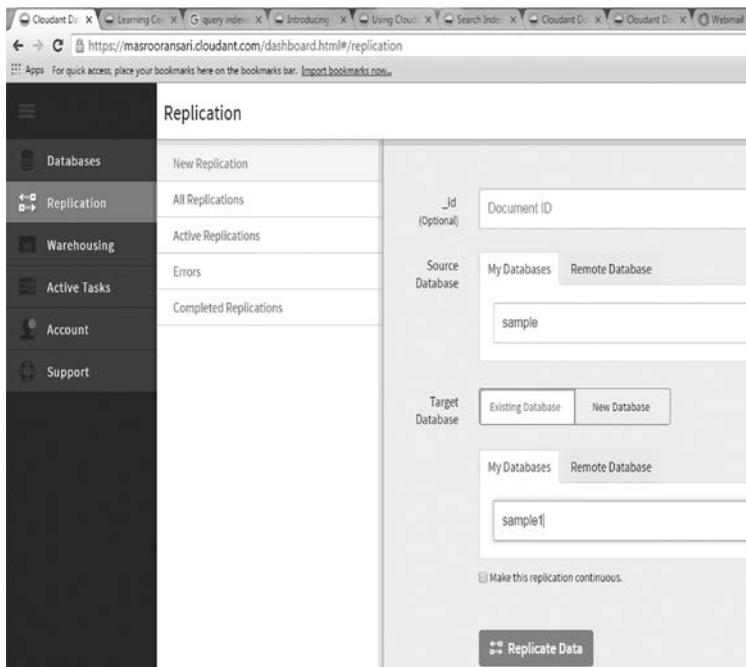
```

Example 2:

```

DELETE / cloudant_db
/_design/sample?rev=2-
ac58d589b37d01c00f45a4418c5a15a8
HTTP/1.1
Content-Type: application/JSON

```

**Finding documents using an index****Method:** POST**URL Path:** /cloudant_db/_find**Response Body:**JSON object representing the query results**Roles permitted:** _reader**List all Cloudant query indexes****Method:** GET**URL Path:** /cloudant_db/_index**Response Body:**JSON object describing the indexes**Roles permitted:** _reader

Step 9: Description of return codes is given below.

Codes	Description
200	Index has been created successfully or already existed
201	Database has been created successfully
202	The database has been successfully created on few nodes, but the number of nodes is less than the write plenum
304	Not modified
400	It is a bad request. The requested body does not have the specified format
401	The item requested by the client was not available using the given authorization, or authorization was not supplied
403	Invalid database name
404	The requested resource could not be found. The structure consists of two keys, error and reason
405	Request was sent utilizing an invalid HTTP request type for the URL requested. For instance, you have requested a PUT while a POST is required, and this type of errors can be occurred by invalid URL strings
406	The requested content type is not supported by the server
412	The request headers sent by the client and the capacities of the server do not match with server
415	Do not support content type
416	Specified range in the sent request header cannot be convinced by the server
417	Do not send documents in bulk
500	The request sent by the client was invalid because submitted JSON or information as a part of the request was invalid
503	Cloudant request could not be processed because request may be misspelled in Cloudant account name

Step 10: List of available operators and arrays used for Cloudant queries:

Operator	Argument	Purpose
\$and	Array	Valid if all the selectors match with the array data
\$or	Array	Valid if any of the selectors match with the array data
\$not	Selector	Valid if the given selectors do not match
\$nor	Array	Valid if none of the selectors match with the array data
\$all	Array	Matches an array value if it consists of all the elements of the argument array
\$elemMatch	Selector	Matches and gives back all documents that contain an array field with at least one element that matches all the predetermined query criteria

References

1. Cloudant (n.d.). Retrieved July 6, 2016, from <https://en.wikipedia.org/wiki/Cloudant>
2. IBM Cloudant (n.d.). Retrieved July 5, 2016, from <http://www-01.ibm.com/software/data/cloudant/>

19

Hands-On InfluxDB

Khaleel Ahmad and Masroor Ansari

CONTENTS

19.1 Introduction	341
19.2 Querying Data Using the HTTP API.....	341
References.....	353

19.1 Introduction

InfluxDB is a time series database developed starting from the earliest stage to manage high write and query loads. It is the second part of the TICK stack. InfluxDB is supposed to be used as a backing store for any use case involving voluminous time-stamped data, Internet of Things sensor data, DevOps monitoring, and real-time analytics.

InfluxDB supports those developers or professionals or researchers who are working with time series data. Customized excessive efficiency data store is written chiefly for time series data. The TSM engine makes it possible for prime ingest speed and data compression. It arranges into a solitary pair without external dependencies. It plug-ins abutment for additional data ingestion protocols, for example, OpenTSDB, Graphite, and Collectd. SQL-like query language is tailored to calmly query aggregated data [1,2].

19.2 Querying Data Using the HTTP API

The HTTP API is the main resource for querying data in InfluxDB.

For executing a query, send a GET request, set the URL parameter `db` as the objective database, and set the URL parameter `q` of query. The example given below uses the HTTP API to query the same database.

```
curl -GET 'http://localhost:8086/query?pretty=true' --data-urlencode "db=mydb" --data-urlencode "q=SELECT value FROM cpu_load_short WHERE region='us-west'"
```

To install InfluxDB on Windows, follow step-by-step process as below [3]:

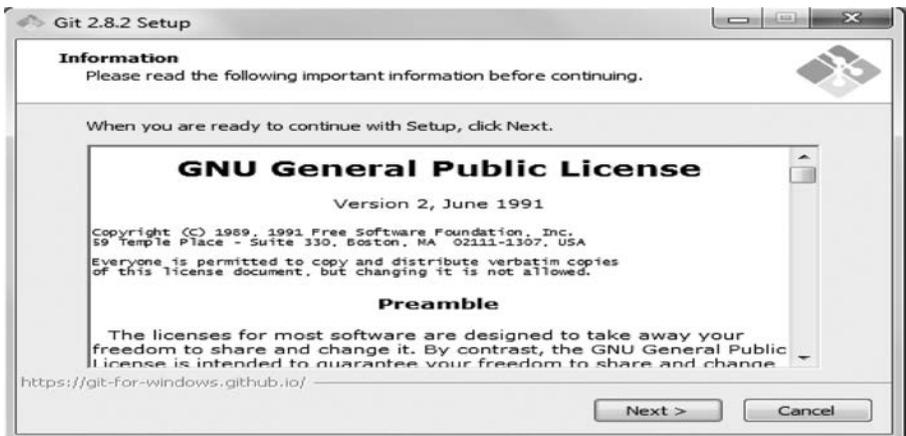
Step 1: Download GIT, and go (Git-2.8.2-64-bit, go1.6.2.windows-amd64) through the following links:

For Git: <https://git-for-windows.github.io/>

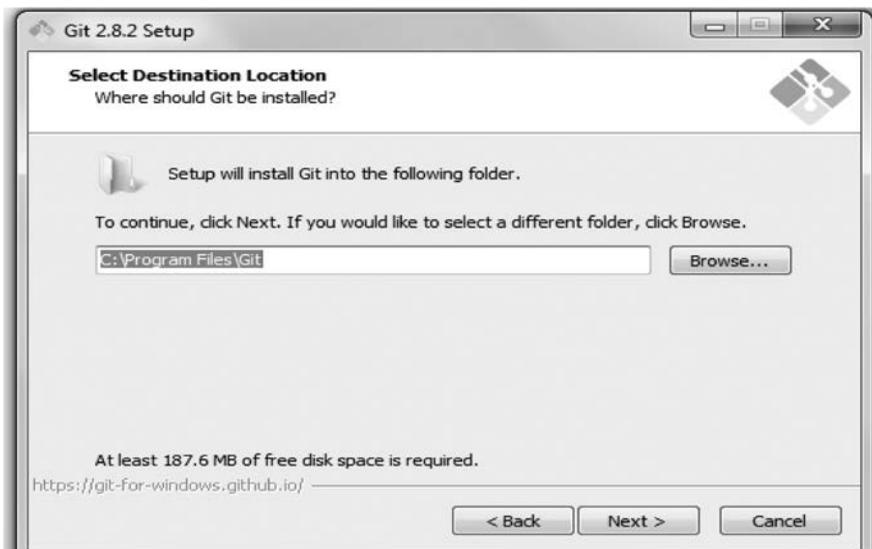
For Go Programming language: <https://golang.org/dl/>

Step 2:

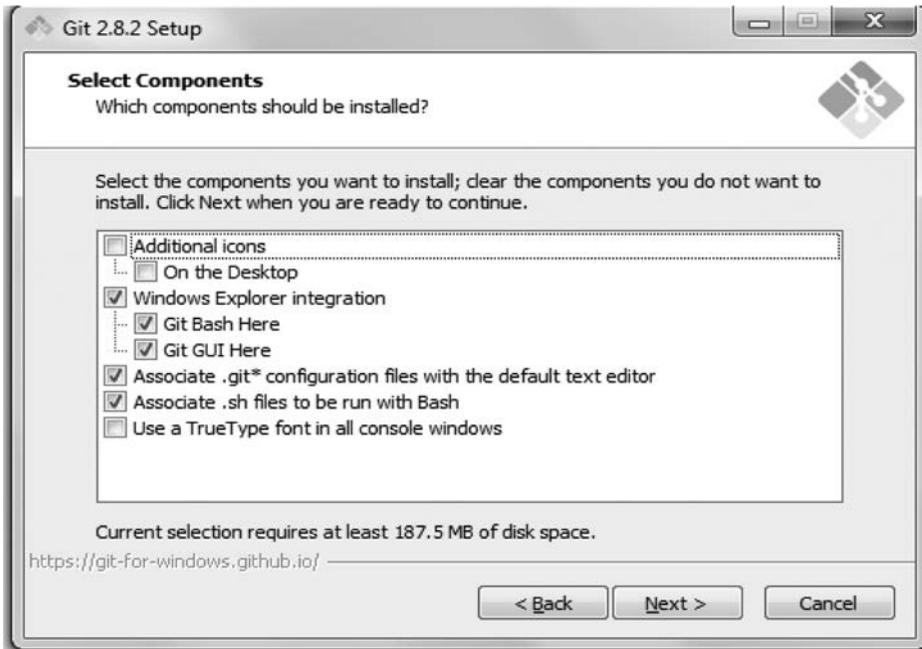
Install Git: Run the application after completion of downloading, a setup will appear, read the GNU General Public License information carefully, and then click on the *Next* button.



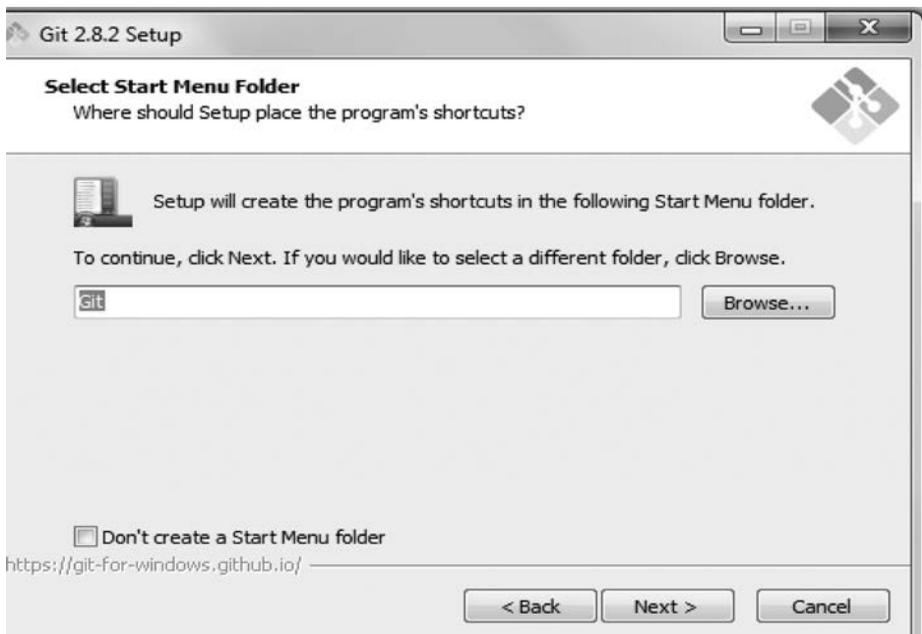
Step 3: Select your destination folder by clicking on the browse button where you want to install the Git, and click on the *Next* button.



Step 4: Select or unselect the component as per your requirements for installation, and then click on the *Next* button.



Step 5: Choose the Start Menu folder to place the program shortcut or check the box to stop the creation of Start Menu folder and click on the *Next* button.



Step 6: Check the box button for Enable file system caching, and then click on the *Next* button.



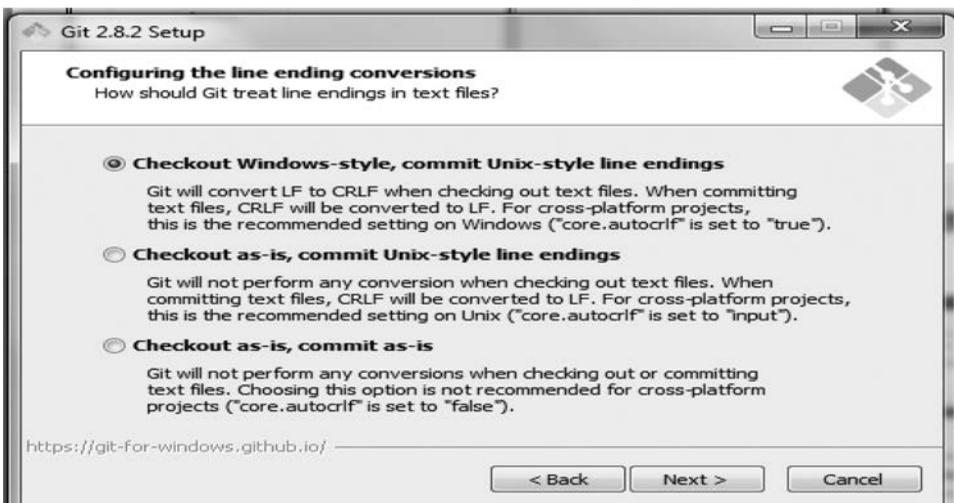
Step 7: Check the radio button for configuring the terminal emulator, and then click on the *Next* button.



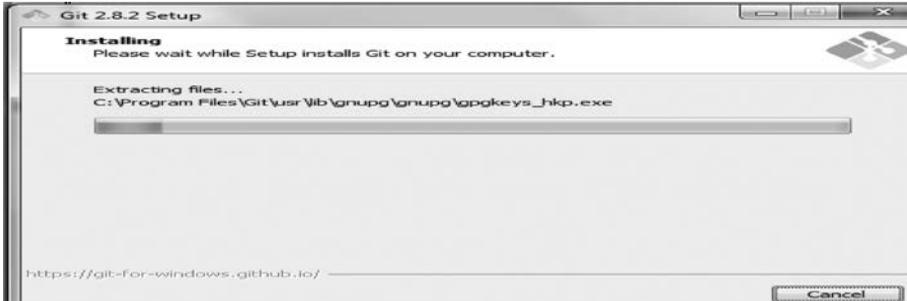
Step 8: Check the radio button based on the needs to adjust the path environment, and then click on the *Next* button.



Step 9: Set the configuration for line ending conversions. Select the Radio button for choosing own style.

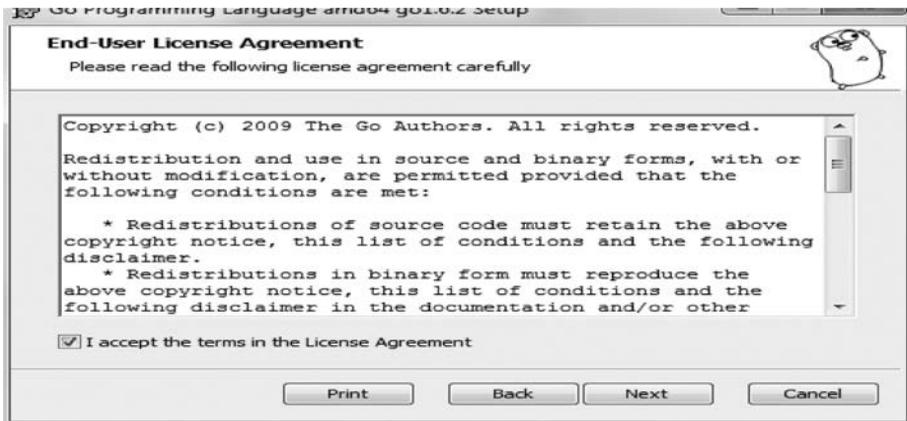


Step 10: Installation will be processed. So, wait until the installation is completed.



Step 11: Install `go1.6.2.windows-amd64`

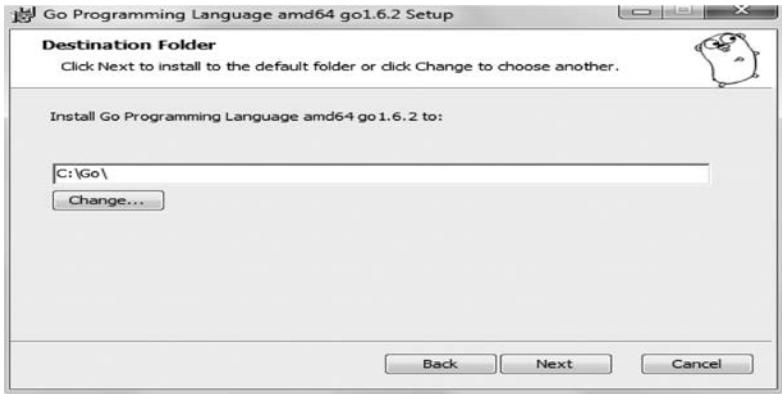
Run the application after completion of downloading, a setup will appear, read the End-User License Agreement carefully, and then click on the *Next* button.



Step 12: Go Programming Language amd64 go1.6.2setup will appear, and then click on the *Next* button.



Step 13: Select your destination folder by clicking on the *change* button where you want to install the GO1.6.2, and click on the *Next* button.



Step 14: Click on the *Install* button.



Step 15: Installation will be processed. So, wait until the installation is completed.



Step 16:

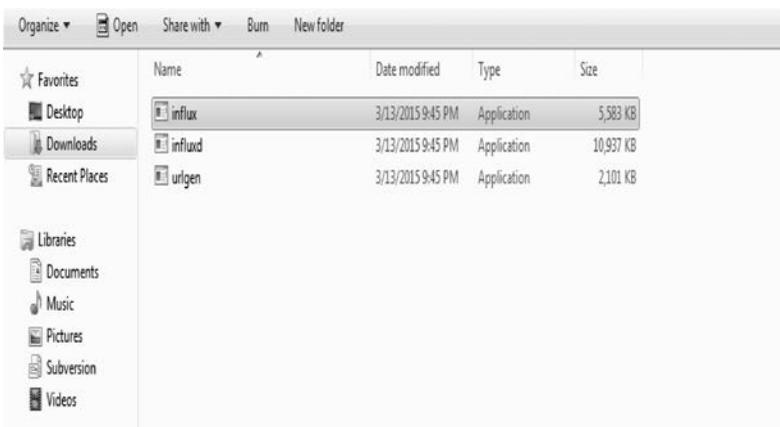
Download InfluxDB from the following link: https://github.com/adriencarbonne/influxdb/releases/download/v0.9.0-rc11/influxdb_v0.9.0-rc11.zip

After completing the installation, three files are generated.

Influx: This file is used for managing the database. Creation of database and manipulation under database can be performed by utilizing this file.

Influxd: It is utilized for formation of server at local host (local machine) so that association with databases are effectively handled. It gives an environment for running influx database on the system.

Urlgen: This file handles execution of queries.



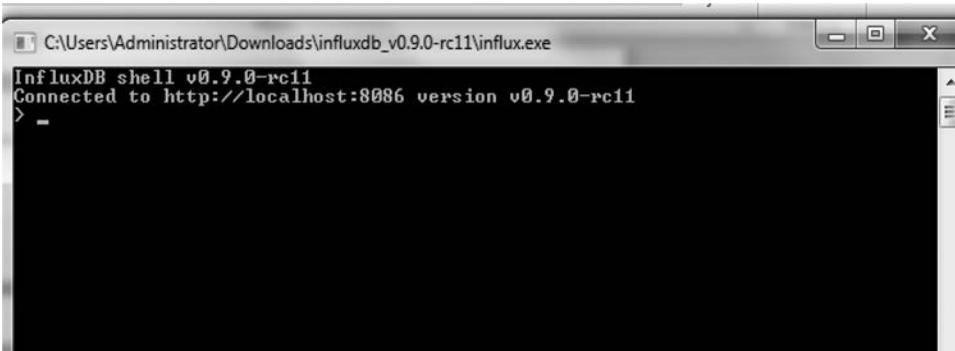
InfluxDB returns JSON. The outcomes of query come into view in the "results" array. If an error occurs, InfluxDB sets an "error" key with an elucidation of the error.

```
{
  "results": [
    {
```

Step 17: Execute *Influxd* file (*Influxd.exe*) for creation of environment for influxDB.

```
C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influxd.exe
[server] 2016/05/19 10:15:29 influxdb started, version v0.9.0-rc11, commit 55c0e4434f60dd64516e35ae266e055356ca1aa4
[raft] 2016/05/19 10:15:29 log open: created at C:\Users\Administrator\influxdb\broker\raft, with ID 1, term 1, last applied index of 18
[raft] 2016/05/19 10:15:29 log open: promoting to leader immediately
[raft] 2016/05/19 10:15:29 log state change: stopped => leader
[raft] 2016/05/19 10:15:29 log state change: leader => leader
[server] 2016/05/19 10:15:29 broker listening on :8086
[server] 2016/05/19 10:15:29 Loading metadata index for andrew_clark
[server] 2016/05/19 10:15:29 Loading metadata index for nydb
[server] 2016/05/19 10:15:29 Loading metadata index for sample
[server] 2016/05/19 10:15:29 broker enforcing retention policies with check interval of 10m0s
[server] 2016/05/19 10:15:29 shard group pre-create with check interval of 45m0s
[server] 2016/05/19 10:15:29 data node #1 listening on :8086
[server] 2016/05/19 10:15:29 starting admin server on :8083
[server] 2016/05/19 10:15:29 Sending anonymous usage statistics to n.influxdb.com
[messaging] 2016/05/19 10:15:30 connected to broker: http://SERVER2-PC:8086/messaging/messages?replicaID=1
[http] 2016/05/19 10:15:34 127.0.0.1 - - [19/May/2016:10:15:34 +0530] GET /ping HTTP/1.1 204 0 - InfluxDBShell/v0.9.0-rc11 85f5111e-1d7c-11e6-91fa-00215a20e22b
[server] 2016/05/19 10:25:30 retention policy enforcement check commencing
[server] 2016/05/19 10:35:30 retention policy enforcement check commencing
[server] 2016/05/19 10:45:30 retention policy enforcement check commencing
```

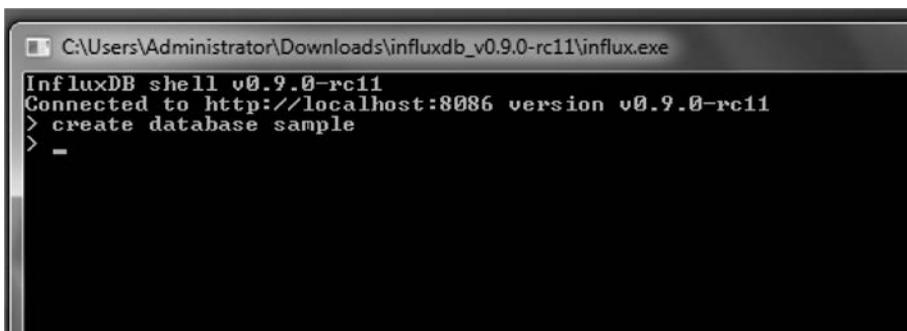
Step 18: For Command Line Interface (CLI), execute *Influx.exe* (*Influx file*) for creation of database.



```
C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> -
```

Step 19: Create the database using query [4].

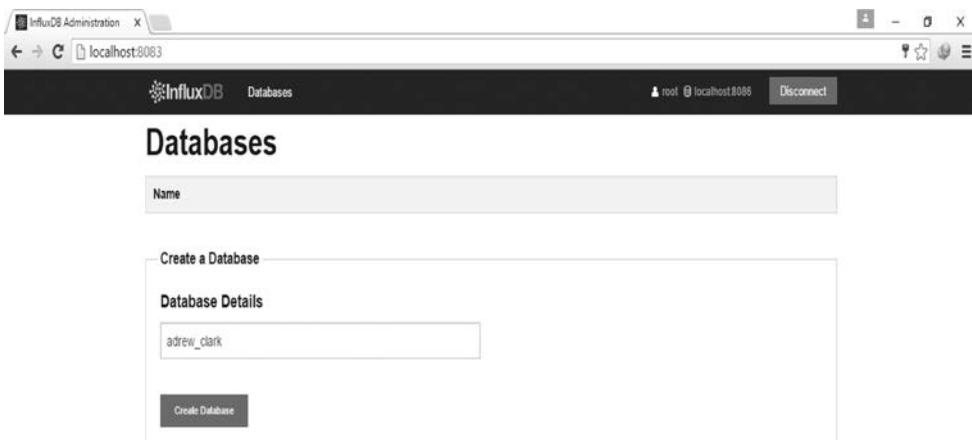
CLI Method: Type `create database sample` on Influx CLI Window.



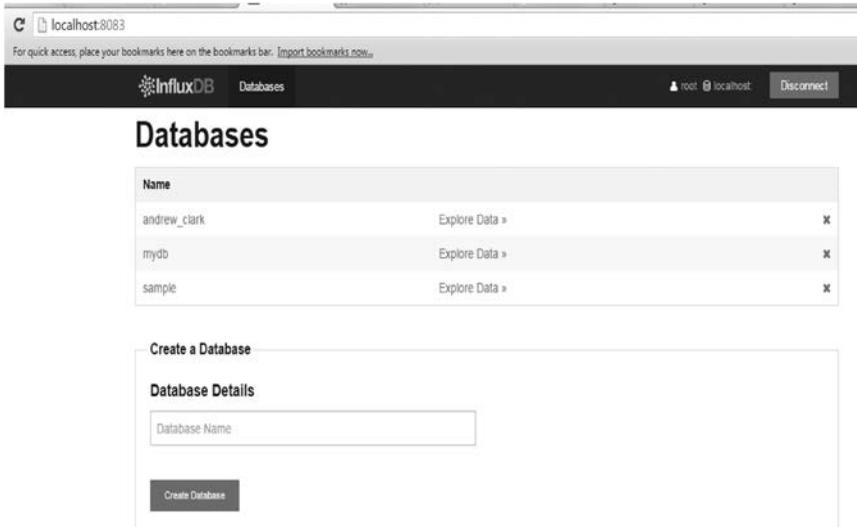
```
C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> create database sample
> -
```

GUI Method: Create the database using GUI.

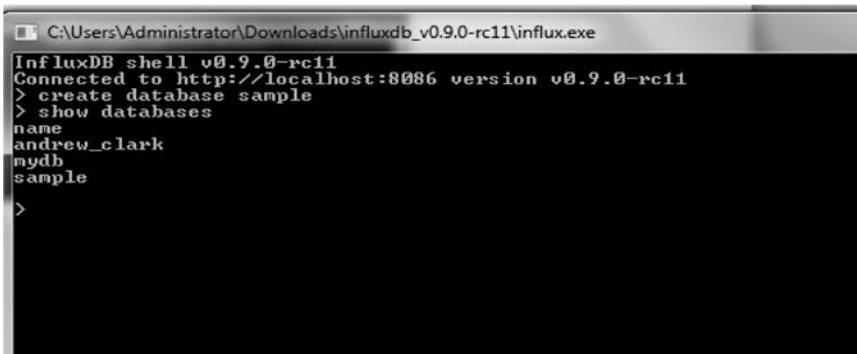
Open the browser, and type “`http://localhost:8083`” in the URL.



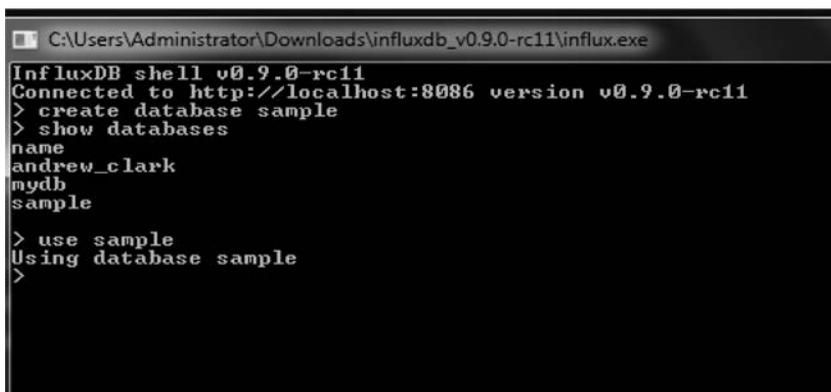
Step 20: Show the created databases using query.



Influx CLI Window: Type `show database`

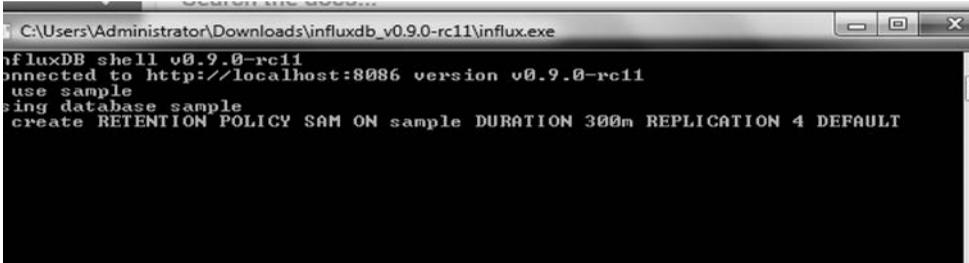


Influx CLI Window: Type `use sample`



Step 21: Create retention policy using query.

Influx CLI Window: Create RETENTION POLICY SAM ON sample DURATION 300 m REPLICATION 4 DEFAULT (Query can be run without default).



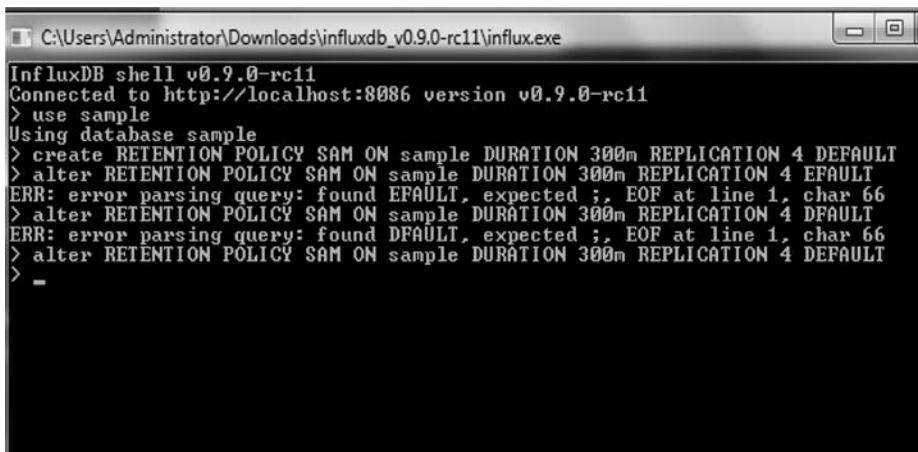
```

C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> use sample
Using database sample
> create RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT

```

Step 22: Alter Retention policy using query.

Influx CLI Window: Alter RETENTION POLICY SAM ON sample DURATION 300 m RELPLICATION 4 DEFAULT.



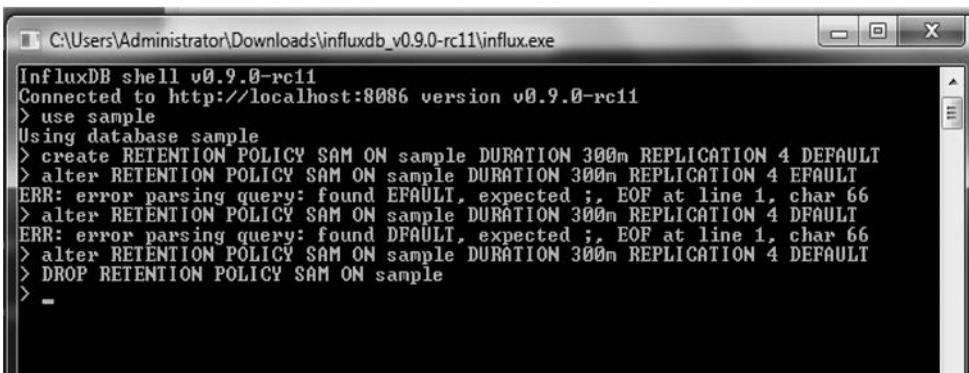
```

C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> use sample
Using database sample
> create RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 EFAULT
ERR: error parsing query: found EFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DFAULT
ERR: error parsing query: found DFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> -

```

Step 23: Drop Retention Policy using query.

Influx CLI Window: DROP RETENTION POLICY SAM ON sample



```

C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> use sample
Using database sample
> create RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 EFAULT
ERR: error parsing query: found EFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DFAULT
ERR: error parsing query: found DFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> DROP RETENTION POLICY SAM ON sample
> -

```

Step 24: Show existing users using query.

Influx CLI Window: Show users.

```

C:\Users\Administrator\Downloads\influxdb_v0.9.0-rc11\influx.exe
InfluxDB shell v0.9.0-rc11
Connected to http://localhost:8086 version v0.9.0-rc11
> use sample
Using database sample
> create RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 EFAULT
ERR: error parsing query: found EFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DFAULT
ERR: error parsing query: found DFAULT, expected ;, EOF at line 1, char 66
> alter RETENTION POLICY SAM ON sample DURATION 300m REPLICATION 4 DEFAULT
> DROP RETENTION POLICY SAM ON sample
> show measurements
> show users
user      admin
> _

```

Step 25: Type of literals used in influx database.

Literals	Format
Query	statement { ; statement }
bool_lit	TRUE FALSE.
regex_lit	"/" { unicode_char } "/"
int_lit	("1" ... "9") {digit }
float_lit	Int_lit "." Int_lit
String_lit	'{Unicode_char}'
Time_lit	"2006-01-02 15:04:05.999999" "2006-01-02"

Step 26: Type of keywords in influx database as follows:

All	Alter	As	Asc	Begin	By
Create	Continuous	Database	Databases	Default	Delete
Desc	Drop	Duration	End	Exists	Explain
Field	From	Grant	Group	If	In
Inner	Insert	Into	Key	Keys	Limit
Show	Measurement	Measurements	Not	Offset	On
Order	Password	Policy	Policies	Privileges	Queries
Query	Read	Replication	Retention	Revoke	Select
Series	Server	Slimit	Soffset	Teg	To
User	Users	Values	Where	With	Write

Step 27: Types of units available for defining parameters in influx database are as follows:

Units	Meaning
u or μ	Microseconds (1 millionth of a second)
ms	Milliseconds (1 thousandth of a second)
s	Second
M	Minute
H	Hour
D	Day
W	Week

References

1. InfluxData Documentation (n.d.). Retrieved June 14, 2016, from <https://docs.influxdata.com/influxdb/v0.13/>
2. InfluxData Documentation (n.d.). Retrieved June 10, 2016, from https://docs.influxdata.com/influxdb/v0.9/introduction/getting_started/
3. [0.9.2] (regression) Error reading from measurement named "Measurement" Issue #3467 influxdata/influxdb (n.d.). Retrieved June 14, 2016, from <https://github.com/influxdata/influxdb/issues/3467>
4. InfluxData Documentation (n.d.). Retrieved June 15, 2016, from https://docs.influxdata.com/influxdb/v0.9/query_language/query_syntax/#identifiers-double-quoted



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

20

Hands-On Redis

Khaleel Ahmad and Mohd Javed

CONTENTS

20.1 Introduction.....	355
20.2 Advantages of Redis.....	356
References.....	363

20.1 Introduction

Redis is a freely available software [Berkeley Software Distribution (BSD) licensed], in-memory data structure store, utilized as database, cache, and message agent. Redis is written in C Language and works in most Portable Operating System Interface for Unix (POSIX) systems like Linux, OS X, and *BSD with no external conditions. OS X and Linux are the two operating systems where Redis is invented and tested. There is no official support for Windows builds; however, Microsoft develops and keeps up a Win-64 port of Redis.

Redis is a key-value database. It is a solution for building elite and adaptable web applications. Not like a relational database, this is constituted of tables that incorporate columns of data definitions and rows of data, and a key-value database that stores value, each of which might be referenced by a unique key. It bolsters data structures, for example, lists, strings, sets, hashes, sorted sets with range queries, hyperloglogs, bit-maps, and geospatial indexes with radius queries. Redis has integrated replication, LRU eviction, transactions, Lua scripting, and distinct levels of on-disk persistence, and it gives high accessibility through Redis Sentinel and automatic segregation with Redis Cluster.

Redis also helps trivial-to-setup master-slave asynchronous replication, with quack non-blocking first synchronization, auto reconnection with half-done resynchronization on net split [1,2].

Other features of Redis include the following:

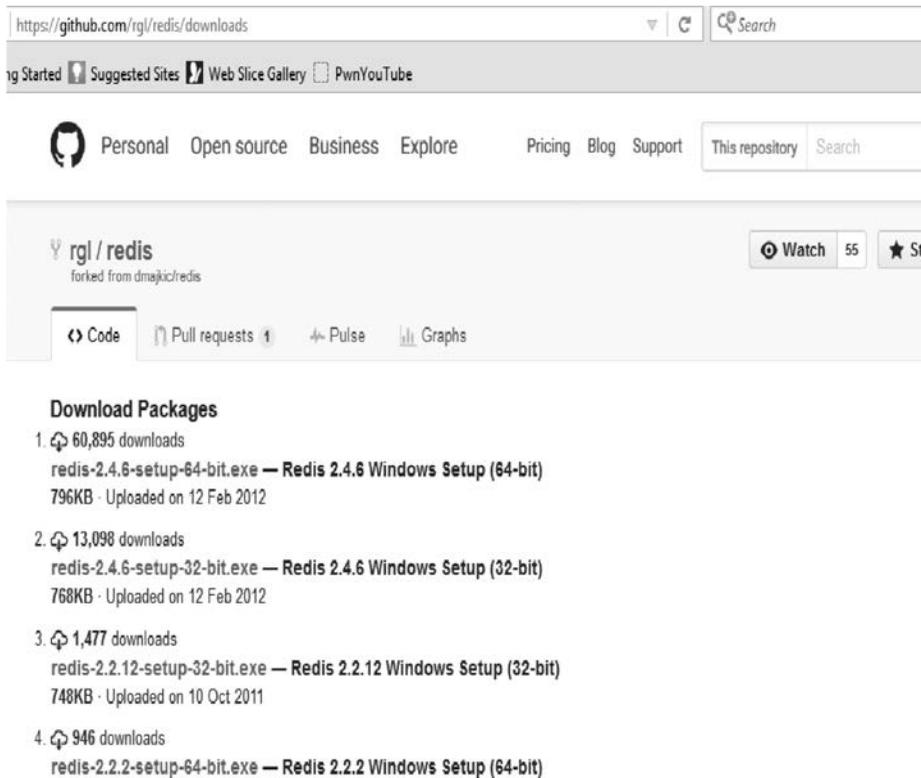
- Publishers/Subscribers (Pub/Sub)
- Keys with a limited time to live
- Automatic failover
- It loads the whole dataset into RAM, so it is fast
- It writes the database into a disk, so it is persistent
- It bolsters master-slave replication, so it is scalable

20.2 Advantages of Redis

- *Supports Rich Data Types*: Redis natively bolsters most of the data types, for example, set, list, sorted set, and hashes. This makes it very effortless to solve a diversity of problems because we know which issue can be managed better by which data type.
- *MultiUtility Tool*: Redis is a multiutility tool and may be utilized in some of use cases like messaging queues (Redis supports Publish/Subscribe), caching, any short-lived data in application, for example, web page hit counts, web application sessions, etc.
- *Exceptionally Fast*: Redis is exceedingly fast and can execute about 110,000 SETs per second and about 81,000 GETs per second.
- *Operations are atomic*: The entire Redis operations are atomic, which assures that if two clients simultaneously access Redis server, they will get the up-to-date value [2].

Step 1: To install *Redis* on Windows, follow step-by-step process:

Download *Redis* through this link: <https://github.com/rgl/redis/downloads>



The screenshot shows the GitHub repository page for `rgl/redis`, which is a fork of `dmajkic/redis`. The page displays the repository name, navigation links (Personal, Open source, Business, Explore, Pricing, Blog, Support), and a search bar. Below the repository name, there are buttons for 'Code', 'Pull requests', 'Pulse', and 'Graphs'. The 'Download Packages' section is highlighted, listing four packages with their download counts, file names, and upload dates:

Rank	Downloads	File Name	Description	Upload Date
1.	60,895	redis-2.4.6-setup-64-bit.exe	Redis 2.4.6 Windows Setup (64-bit)	12 Feb 2012
2.	13,098	redis-2.4.6-setup-32-bit.exe	Redis 2.4.6 Windows Setup (32-bit)	12 Feb 2012
3.	1,477	redis-2.2.12-setup-32-bit.exe	Redis 2.2.12 Windows Setup (32-bit)	10 Oct 2011
4.	946	redis-2.2.2-setup-64-bit.exe	Redis 2.2.2 Windows Setup (64-bit)	-

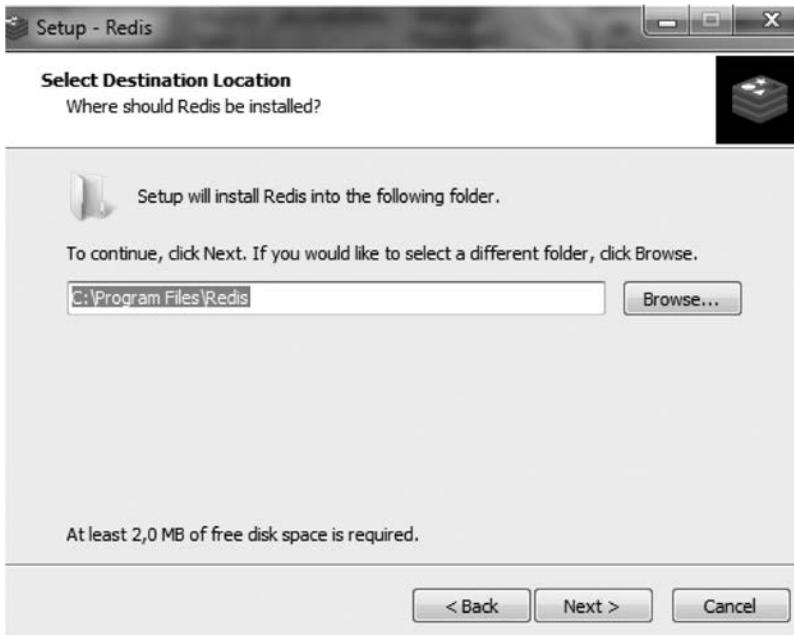
Step 2: Run the application after completion of downloading, *Redis setup Wizard* screen will appear, and then click on the *Next* button.



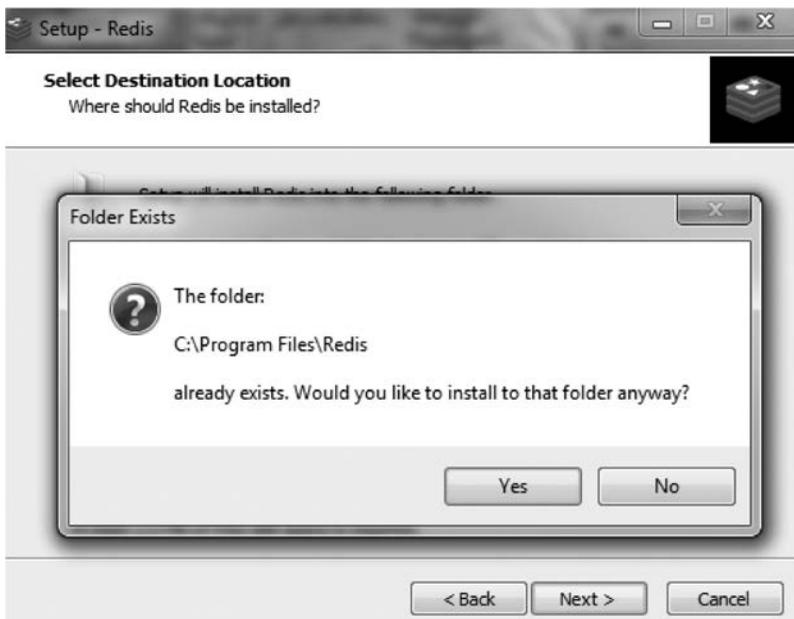
Step 3: License Agreement screen will appear, select *Accept Agreement* radio button, and then click on the *Next* button.



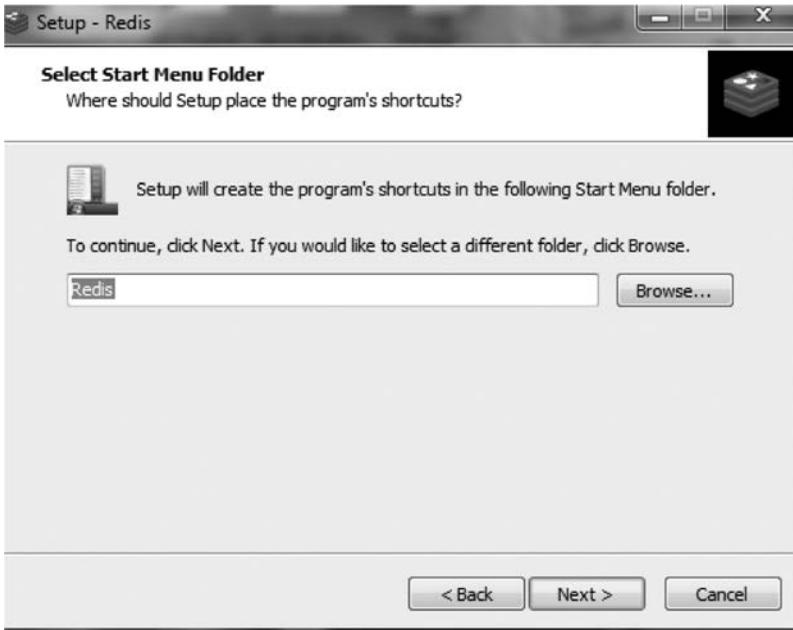
Step 4: Select your destination folder by clicking on the browse button where you want to install the *Redis* and click on the *Next* button.



Step 5: Click on the *Yes* button.



Step 6: Click on the *Browse* button to place the program's shortcut in a different folder. If not so, click on the *Next* button.



Step 7: Here ready to install Redis, and you just click the install button.



Step 8: After completing the installation process, click on the *Finish* button to finish the setup.



Step 9: After finishing the *Redis* setup, *README notepad* file will appear on the screen.

```

File Edit Format View Help
About this fork
-----
This fork lets you run Redis Server as a Windows service.

It introduces the redis-service.exe binary; this is a Windows
service that
simply starts and stops the redis-server.exe binary.

It expects to find redis-server.exe and conf\redis.conf files in
the same
directory as redis-service.exe.

This fork also introduces a Windows installer; this will:

* install redis into Program Files.
* configure redis to store its database dumps inside the
"data"
subdirectory, and logs inside "logs" subdirectory.
* create the RedisService Windows account (with Logon as
service
privilege)
* install a Windows Service to automatically start Redis

```

Step 10: Now, we will check if Redis server is working or not. Open *Redis Client*, type command *PING*, and press enter.

```

Redis Client
redis 127.0.0.1:6379> ping
PONG
redis 127.0.0.1:6379> _

```

Redis server is giving respond of PING command. It means that *Redis server* is installed successfully in your system.

Step 11: Now, we store some values in Redis server, type `set firstnumber 1` and press enter.



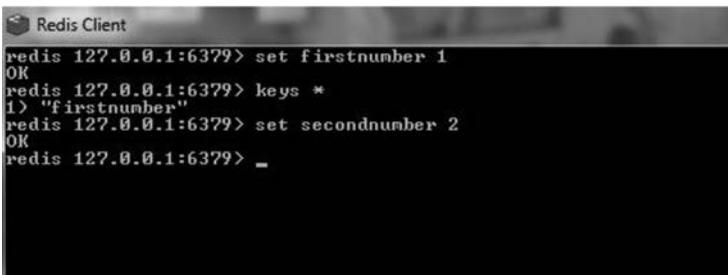
```
Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379>
```

Step 12: If you want to know that how many keys are stored in the *Redis* sever, type `keys *` and press enter.



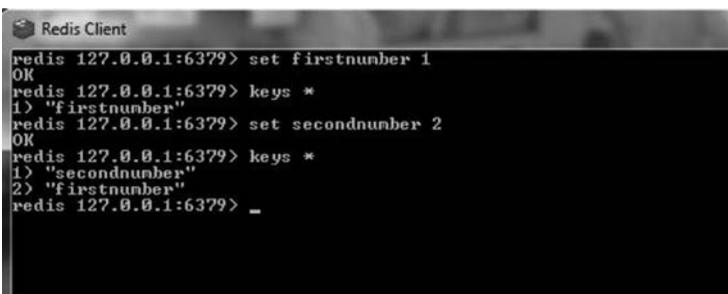
```
Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379>
```

Step 13: For storing one more string number in the Redis server, type `set secondnumber 2` and press enter.



```
Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379> set secondnumber 2
OK
redis 127.0.0.1:6379> _
```

Step 14: If you want to know that how many keys are stored in the *Redis* sever, type `keys *` and press enter.



```
Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379> set secondnumber 2
OK
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> _
```

Step 15: If you want to delete all the string from the *Redis server*, type `flushall` and press enter.

```

Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379> set secondnumber 2
OK
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> flushall
OK
redis 127.0.0.1:6379> _

```

Step 16: Now, we can check that how many keys are stored in the *Redis server*, type `keys *` and press enter.

```

Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379> set secondnumber 2
OK
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> flushall
OK
redis 127.0.0.1:6379> keys *
(empty list or set)
redis 127.0.0.1:6379>

```

It shows “empty list or set,” which means that there is no any type of data in Redis server.

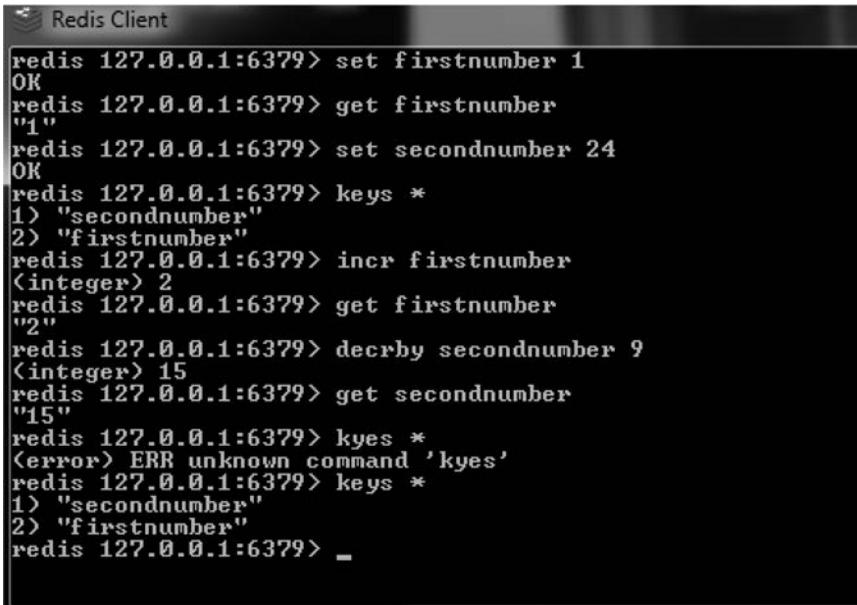
Step 17: If you want to again store the values in the *Redis server*, type `setfirstnumber 1`. If you want to get the first number, type `get firstnumber`.

```

Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> keys *
1) "firstnumber"
redis 127.0.0.1:6379> set secondnumber 2
OK
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> flushall
OK
redis 127.0.0.1:6379> keys *
(empty list or set)
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> get firstnumber
"1"
redis 127.0.0.1:6379>

```

Step 18: Now, if you want increment and decrement values, type `incr firstnumber` and `decrby secondnumber`, respectively.



```
Redis Client
redis 127.0.0.1:6379> set firstnumber 1
OK
redis 127.0.0.1:6379> get firstnumber
"1"
redis 127.0.0.1:6379> set secondnumber 24
OK
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> incr firstnumber
(integer) 2
redis 127.0.0.1:6379> get firstnumber
"2"
redis 127.0.0.1:6379> decrby secondnumber 9
(integer) 15
redis 127.0.0.1:6379> get secondnumber
"15"
redis 127.0.0.1:6379> kyes *
(error) ERR unknown command 'kyes'
redis 127.0.0.1:6379> keys *
1) "secondnumber"
2) "firstnumber"
redis 127.0.0.1:6379> _
```

References

1. Redis (n.d.). Retrieved June 15, 2016, from <https://en.wikipedia.org/wiki/Redis>
2. Redis Overview (n.d.). Retrieved June 5, 2016, from http://www.tutorialspoint.com/redis/redis_overview.htm



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

21

Hands-On RethinkDB

Dildar Hussain and Khaleel Ahmad

CONTENTS

21.1 Introduction.....	365
References.....	375

21.1 Introduction

RethinkDB is the first open source which is a scalable JSON database built starting with the more basic thinks for the real-time web. RethinkDB is the first open-source, scalable database designed specifically to push data to applications in real time. RethinkDB's real-time push architecture in an exciting and impressive way reduces the time and effort which is essential to build scalable real-time applications. The RethinkDB server is written in C++ and runs on 32- and 64-bit Linux systems, as well as OS X 10.7 and above. The RethinkDB server is licensed under the GNU Affero General Public License v3.0. Client drivers can run on any platform where their languages are supported. Moreover, RethinkDB proffers a flexible query language, intuitive operations, and monitoring APIs, and it is straightforward to set up and study.

Modern applications have the need of sending data directly to the client in real time. The query-response database gets right of entry to model works glowing on the web applications because it redirects to HTTP's request-response. For example, while a user changes the location of a button in a collaborative design application, which means various users working on same application, the server has to alert other users that are concurrently working on the same project. Web browsers support these use cases via Web Sockets and long-lived HTTP connections, but adapting database systems to real-time needs still presents a huge engineering challenge.

The RethinkDB architecture is called as change feed architecture that is designed to allow every client to open many real-time feeds. Since modern web and mobile applications often have tens of thousands of simultaneous clients, RethinkDB's feeds are designed to be tremendously scalable. With the aid of default, no write is always acknowledged until it is safely committed to disk.

Users never have to worry about sending queries to definite nodes. Connect your clients to any node in a cluster, and queries might be routed to the best possible destination. Sophisticated queries such as joins and filters will be executed simultaneously, with outcome recombined and streamed back to the client. Everything happens automatically behind the scenes [1,2].

Step 1: Installation process of RethinkDB

Use the below given link to select operating system supported by RethinkDB:
<https://rethinkdb.com/docs/install/>

Step 2: Download RethinkDB from below given link:

<https://download.rethinkdb.com/windows/rethinkdb-2.3.4.zip>

Click on *Rethinkdb.exe* file to install the RethinkDB.

Step 3: After installation, open the command prompt.

Type `cd RethinkDB` and press enter.

Type `RethinkDB.exe` and press enter.

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\S>cd RethinkDB
C:\Users\S\RethinkDB>RethinkDB.exe_

```

Step 4: When the server is ready, and then type command “`RethinkDB - -bind all`” on command prompt after closing the first command prompt.

```

C:\Windows\system32\cmd.exe - rethinkdb --bind all
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\S>rethinkdb --bind all
Running rethinkdb 2.3.2-windows-beta (MSC 190023506)...
Running on 6.1.7601 (Windows 7, Server 2008 R2)
Loading data from directory C:\Users\S\rethinkdb_data
Listening for intracluster connections on port 29015
Listening for client driver connections on port 28015
Listening for administrative HTTP connections on port 8080
Listening on cluster addresses: 127.0.0.1, 192.168.137.1, 10.100.102.83, fe80::eccb:742d:308d:61
Listening on driver addresses: 127.0.0.1, 192.168.137.1, 10.100.102.83, fe80::eccb:742d:308d:6b
Listening on http addresses: 127.0.0.1, 192.168.137.1, 10.100.102.83, fe80::eccb:742d:308d:6ba6
Server ready, "J_PC_q68" 7320bc19-1515-4df5-99bf-4d07391735ab

```

Step 5: To install RethinkDB client drivers

To install different types of drivers for clients use below given link: <https://www.rethinkdb.com/docs/install-drivers/>

Step 6: RethinkDB command-line options (<https://rethinkdb.com/docs/cli-options/>)

These options can be passed to the RethinkDB server on the command line on startup.

File path option:

`-d [--directory] path:` Specify directory to store data and metadata

Server name option:

`-n [server -name] arg:` The name for this server will be appeared in the metadata. If not specified, it will be arbitrarily selected from a short list of names.

-t[--server -tag] arg: A tag for this server can be specified various times.

Network option:

--bind{all | address}: Add the address of a local interface to listen on when accepting connections; loopback addresses are enabled by default.

--bind{ http | address }: Bind the web administration UI port to a specific address.

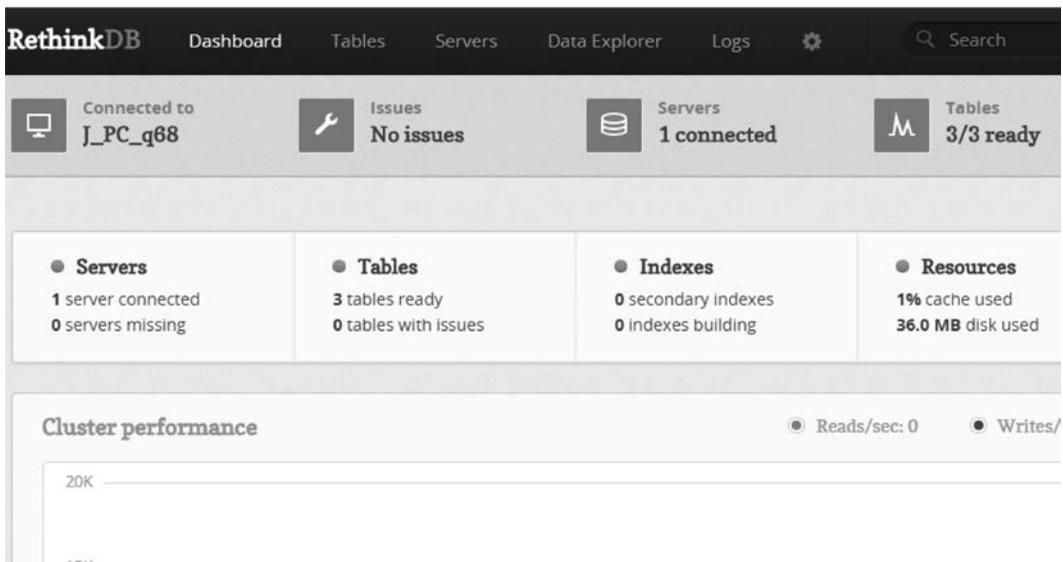
--bind{ cluster | address }: Bind the cluster connection port to a specific address.

Step 7: Creating database and tables by using web browser

When the server is ready, you will open the web browser and type “localhost: 8080” in the URL. This will provide database and table creation platform. In the SQL database, the command prompt is used for creation of database and table but at the place of it is providing GUI-based web browser interface which provides different types of information such as Dashboard, tables, servers, and data explorers.

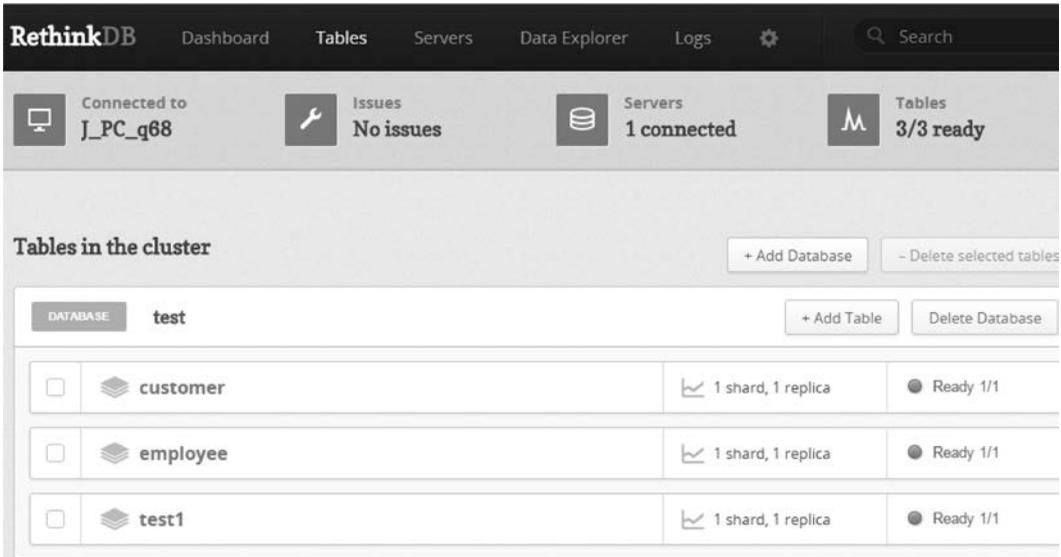
Dashboard: Dashboard provides mainly four types of details as follows:

- *Servers:* How many servers are connected?
- *Tables:* How many tables are created?
- *Indexes:* How many indexes are built?
- *Resources:* How many resources are used for running this application?



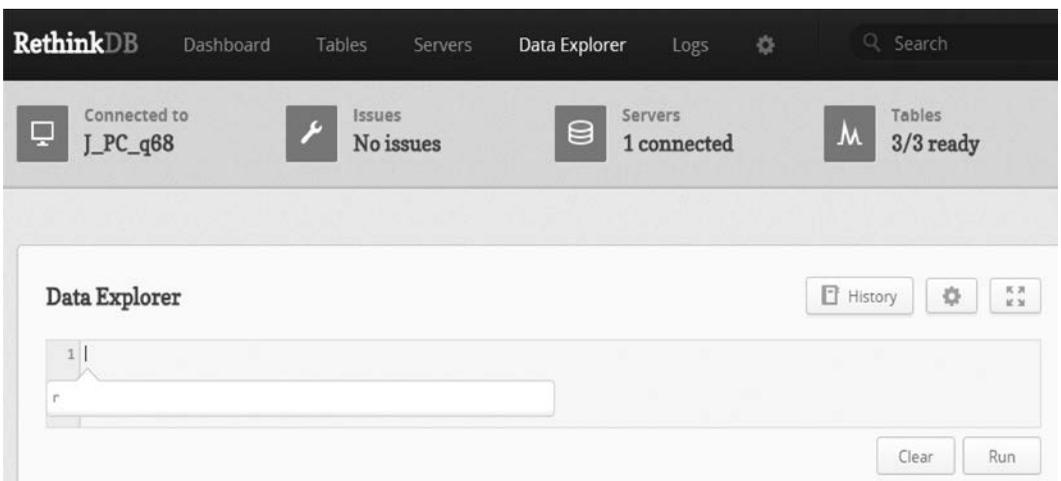
Step 8:

Tables: It will provide the created database and tables details as well as provide buttons for creating database and tables.



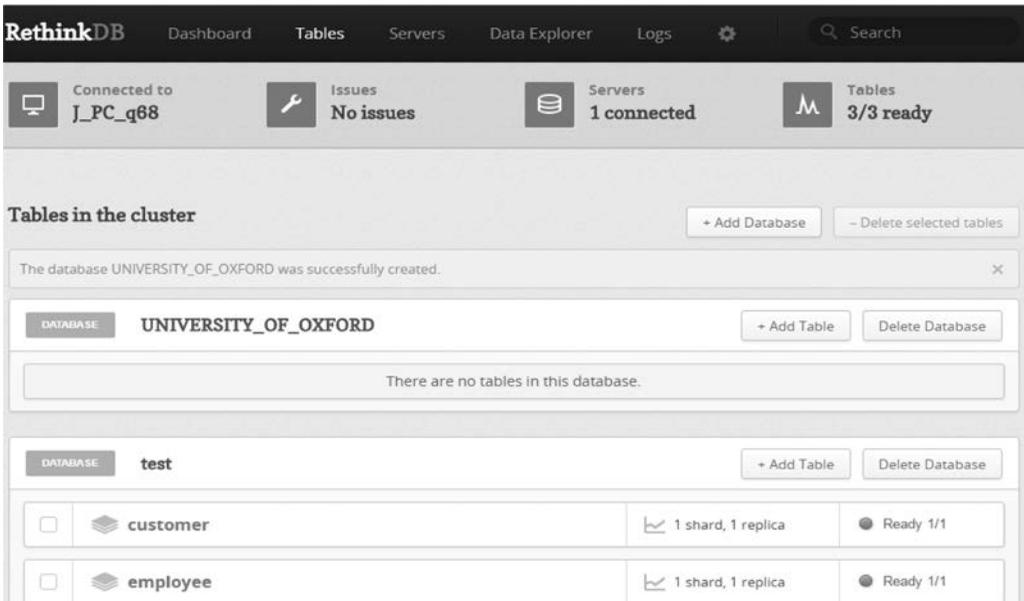
Step 9:

Data explorer: There are various operations of table available, viz. creation, insertion, update, filter, and arranging.



Step 10:

Database creation: Click on "Add Database" button, and then write the database name.

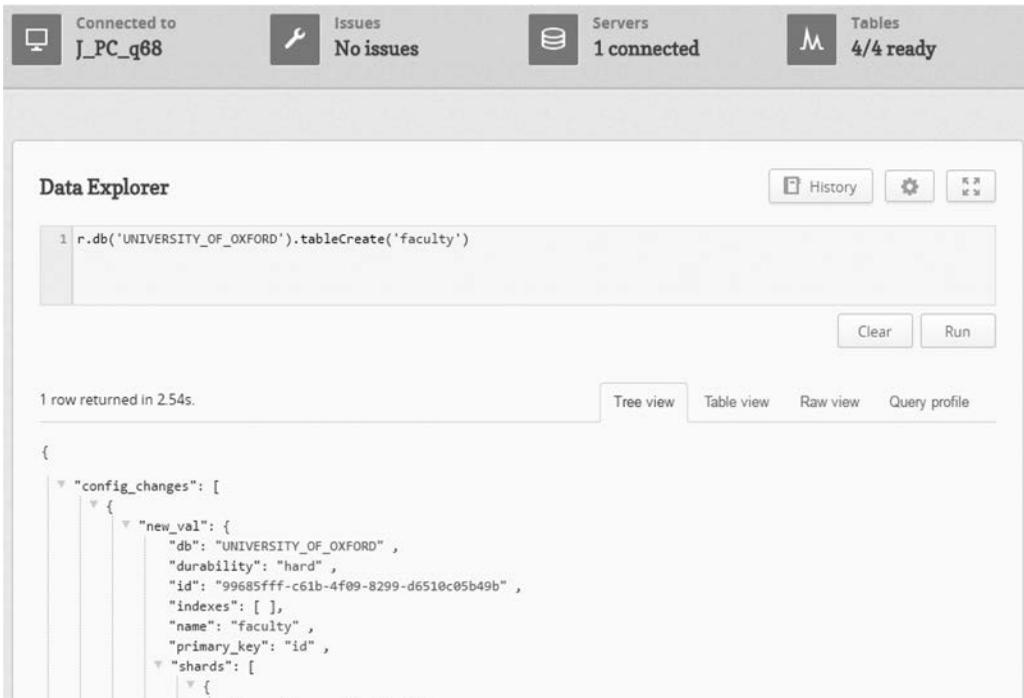


Step 11:

Table creation: Click on Dashboard from menu then write

```
r.db('UNIVERSITY _ OF _ OXFORD').tableCreate('faculty')
```

Where "r" is the top-level ReQL namespace.



Step 12:

Single data insertion into table:

The screenshot shows the Data Explorer interface. At the top, there are status indicators: 'Connected to J_PC_q68', 'Issues No issues', 'Servers 1 connected', and 'Tables 4/4 ready'. The main area is titled 'Data Explorer' and contains a query editor with the following SQL query:

```
1 r.db('UNIVERSITY_OF_OXFORD').table('faculty').insert({"f_id":"a201","f_name":"Jhon","f_age":35})
```

Below the query editor, it says '1 row returned in 172ms.' and provides view options: 'Tree view', 'Table view', 'Raw view', and 'Query profi'. The result is displayed in a JSON format:

```
{
  "deleted": 0 ,
  "errors": 0 ,
  "generated_keys": [
    "a01e74b3-4eb5-4c48-9a27-7cd2f325e0f6"
  ],
  "inserted": 1 ,
  "replaced": 0 ,
  "skipped": 0 ,
  "unchanged": 0
}
```

Multiple data insertion into table:

The screenshot shows the Data Explorer interface. At the top, there are status indicators: 'Connected to J_PC_q68', 'Issues No issues', 'Servers 1 connected', and 'Tables 4/4 ready'. The main area is titled 'Data Explorer' and contains a query editor with the following SQL query:

```
1 r.db('UNIVERSITY_OF_OXFORD').table('faculty').insert([
2 {
3   "f_id":"a202","f_name":"clark","f_age":29
4 },
5
6 {
7   "f_id":"a203","f_name":"andrew","f_age":34
8 },
9
10 {
11   "f_id":"204","f_name":"simon","f_age":39
12 }
13 ])
```

Below the query editor, it says '1 row returned in 102ms.' and provides view options: 'Tree view', 'Table view', 'Raw view', and 'Query profi'. The result is displayed in a JSON format:

```
{
  "deleted": 0 ,
  "errors": 0 ,
  "generated_keys": [
    "ed15489b-1f9e-47b0-b7ce-f2f278943d2f" ,
    "20f60ae4-5c12-4d62-a507-ac7a47ce8f21" ,
    "9f6968a8-a653-4921-a74d-11027e0b0736"
  ],
  "inserted": 3 ,
  "replaced": 0 ,
  "skipped": 0 ,
  "unchanged": 0
}
```

Step 13:

Selection of all data from table:

Data Explorer History Settings

```
1 r.db('UNIVERSITY_OF_OXFORD').table('faculty')
```

Clear

4 rows returned. Displaying rows 1-4 Tree view Table view Raw view Query p

```
{
  "f_age": 29 ,
  "f_id": "a202" ,
  "f_name": "clark" ,
  "id": "ed15489b-1f9e-47b0-b7ce-f2f278943d2f"
}
{
  "f_age": 34 ,
  "f_id": "a203" ,
  "f_name": "andrew" ,
  "id": "20f60ae4-5c12-4d62-a507-ac7a47ce8f21"
}
{
  "f_age": 39 ,
  "f_id": "204" ,
  "f_name": "simon" ,
  "id": "9f6968a8-a653-4921-a74d-11027e0b0736"
}
```

Step 14:

Update of table:

Connected to J_PC_q68 Issues No issues Servers 1 connected Tables 4/4 ready

Data Explorer History Settings

```
1 r.db('UNIVERSITY_OF_OXFORD').table('faculty').filter(r.row("f_age").lt(34)).update({f_age:50})
```

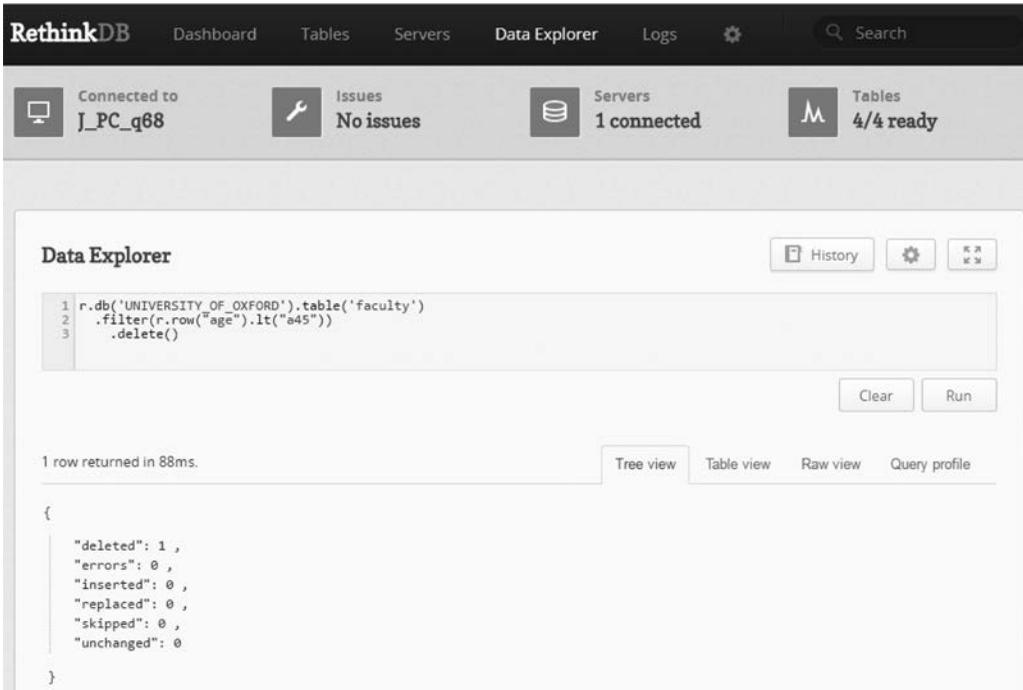
Clear Run

1 row returned in 139ms. Tree view Table view Raw view Query profile

```
{
  "deleted": 0 ,
  "errors": 0 ,
  "inserted": 0 ,
  "replaced": 1 ,
  "skipped": 0 ,
  "unchanged": 0
}
```

Step 15:

Deletion of row from table:



Step 16:

Application program interface using java script:

- Open a connection using the default host and port, specifying the default database.

```
r.connect ({db: 'university_of_oxford'}, function (err, conn)
{
//...
})
```
- Open a new connection to the database.

```
r.connect ({host: 'localhost' port: 28015 db: 'university_of_oxford'},
function (err, conn)
{
//...
}
```

- Open a new connection to the database, specifying a user/password combination for authentication.

```
r.connect ({host: 'localhost' port: 28015 db: 'university_of_oxford' user: 'scott' password: 'tiger' },
function (err, conn)
{
//...
```

- Change the default database on this connection.

```
Conn.use('university_of_oxford')
r.table('faculty').run(conn, ...)
```

- Close an open connection immediately.

```
Conn.close({noreplyWait: false},function(err){if(err) throw err;})
```

Step 17:

Application program interface using python:

- Open a connection using the default host and port, specifying the default database.

```
Conn=r.connect(db= 'university_of_oxford')
```

- Open a new connection to the database.

```
Conn=r.connect (host= 'localhost',
                Port=28015,
                Db= 'university_of_oxford')
```

- Open a new connection to the database, specifying a username/password combination for authentication.

```
Conn=r.connect(host= 'localhost',
                Port=28015,
                Db= 'university_of_oxford'
                User= 'scott'
                Password= 'tiger')
```

- Reply

```
Conn=r.connect(:db=> 'university_of_oxford ').repl()
r.table('faculty').run()
```

- Close the connection

```
Conn.close()
```

- Run

```
r.db('university_of_oxford').table('faculty').tableinsert({"f_id":
"a201", "f_name":"wilson", "f_age":51}).run(conn, norely=true,
durability= 'soft')
```

Step 18:

Application program interface using ruby:

- Open a connection using the default host and port, specifying the default database.

```
Conn=r.connect(:db=>'university_of_oxford')
```

- Open a new connection to the database.

```
Conn=r.connect(:host=> 'localhost',
               :Port=>28015,
               :Db=> 'university_of_oxford')
```

- Open a new connection to the database, specifying a username and password combination for authentication.

```
Conn=r.connect(:host=> 'localhost',
               :Port=>28015,
               :Db=> 'university_of_oxford'
               :User=> 'scott'
               :Password=> 'tiger')
```

- Reply

```
Conn=r.connect(:db=> 'university_of_oxford ').repl
r.table('faculty').run
```

- Close the connection

```
Conn.close
```

- Run

```
r.db('university_of_oxford').table('faculty').tableinsert({"f_id"=>
"a201", "f_name"=>"wilson", "f_age"=>51}).run(conn, norely=>true,
durability=> 'soft')
```

Step 19:

Application program interface using java:

- Open a connection using the default host and port, specifying the default database.

```
Connection conn = r.connection().connect();
```

- Open a new connection, specifying parameters.

```
Connection conn = r.connection()  
.hostname("localhost")  
.Port(28015)  
.dbname("university_of_oxford").connect();
```

- Open a new connection, specifying a user/password combination for authentication.

```
Connection conn = r.connection()  
.hostname("localhost")  
.Port(28015)  
.user("scott", "tiger")  
.dbname("university_of_oxford").connect();
```

- Close an open connection, waiting for no reply writes to finish.

```
Conn.close();
```

- Close an open connection immediately.

```
Conn.close(false);
```

References

1. Advancing the Realtime Web (n.d.). Retrieved June 14, 2016, from <https://rethinkdb.com/blog/realtime-web>
2. Getting Started (n.d.). Retrieved June 16, 2016, from <https://rethinkdb.com/faq/>



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

22

Hands on Neo4j: Graphs for Real-World Applications

Siddhartha Duggirala

CONTENTS

22.1	Introduction to Graphs.....	378
22.1.1	What are Graphs?.....	378
22.1.2	When and Where Do We Encounter Them?.....	378
22.1.3	Graphs versus Other Data Models.....	379
22.2	Use Case: Social Contacts.....	382
22.3	Introduction to Neo4J Graph Database.....	382
22.3.1	Data Types and Data Access Mechanisms Supported.....	383
22.3.2	Cypher Query Language.....	383
22.3.3	Match.....	384
22.3.3.1	RETURN.....	385
22.3.4	Other Cypher Clauses.....	385
22.3.5	Relational Data Model for Social Contacts Application.....	385
22.3.6	Graph Modeling for Social Contacts Application.....	386
22.3.7	Testing the Model.....	388
22.3.8	Delete Command.....	389
22.3.9	Loading Large Amounts of Data from CSV.....	389
22.4	Creating Social Contacts Graph.....	389
22.4.1	Load Books from CSV.....	389
22.4.1.1	Create Books.....	389
22.4.1.2	Book Author Relationship.....	389
22.4.1.3	Book Like Relationship.....	390
22.4.1.4	Person Likes Place.....	390
22.4.1.5	Person Lives Place.....	390
22.4.2	Querying the Database.....	390
22.4.3	Indexes and Constrains.....	391
22.4.4	Finding Information Patterns.....	391
22.5	Common Use Cases.....	392
22.5.1	Social Network.....	393
22.5.2	Recommendation Engines.....	393
22.5.3	Geospatial Data.....	393
22.5.4	Graph Search.....	394
22.5.5	Fraud Analytics.....	394
	Appendix.....	397
	References.....	398

22.1 Introduction to Graphs

22.1.1 What are Graphs?

In the most banal form, graphs are collections of vertices (nodes or points) and the relationships that exist between these vertices called edges. Each vertex can represent items such as person, books, place, or a website, while the edges connecting these nodes can be the relationships between the items, for example, in persons like father, friend, colleague, or authored by, birth place.

The vertices or nodes can be related to each other through directed or undirected edges. Two edges are said to be adjacent if they share a common node. Similarly, two vertices are adjacent if they share a common edge, as shown in [Figure 22.1](#). Graph theory is not a new concept; in fact, it was pioneered by Euler in the eighteenth century. Only in the recent years, the graph theory is being applied to computer science problems.

22.1.2 When and Where Do We Encounter Them?

We use graphs in many walks of our lives often not realizing it. When we want someone to introduce us to a CEO of a big firm, we tap into our friends and colleagues. While brainstorming the ideas within teams, we create a mind map of the ideas. While managing a big project, the project lead creates a dependence graph of resources, timelines, etc. While using maps or the shortest route to our offices or homes, we are using graphs.

We use graphs to describe network flows in pipelines or data flows in datacenters. While architecting IT infrastructures, we will be creating a map of all the applications, servers, databases, and load balancers to effectively manage and provide resources. In social network analysis, we use graph to study relationships between people. Perhaps, the most visible way we use some form of graph every day is social networks such as Facebook, Twitter, and LinkedIn. We query these social graphs to get job referrals, find out people we can collaborate with, or even go to a movie with.

Graphs can be useful in understanding physical movements and processes. Likewise, in biology, to trace the migration paths of certain species or track the spreading of certain diseases, graphs are used.

Gone are the days of information silos, the present and the future are in leveraging complex dynamic relationships in highly connected data. Graph databases address this core issue generating insight, thereby competitive advantages. The ability to understand and analyze large amounts of connected data will determine which companies will be winners or losers. The ability to understand relationships between singers and their audience, customers and their shopping patterns, or logistics and delivery times forms the key competitive advantages for the future-proof companies and user experience as a whole.

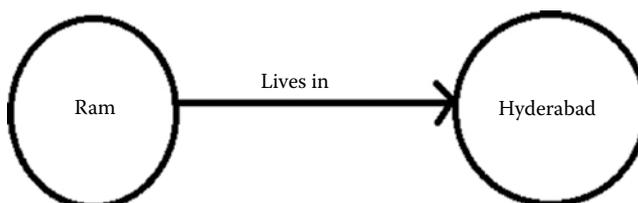


FIGURE 22.1
Simple directed graph.

Graph databases are highly suitable to represent and query highly connected data. The value from connected data is derived by understanding the ways the different entities are related. And for this, we need to name and quantify the connections. The internet titans such as Google, LinkedIn, Twitter, and Amazon realized the potential of graphs and built their proprietary graph systems. With rapid proliferation of graph technologies and open-source options, now even startups or individual developers can benefit from leveraging the graph data without any need to invest heavy resources on graph technologies.

To define formally, a graph database is an online DBMS which supports create, read, update, and delete (CRUD) operations. These are generally built for usage in transaction systems and hence optimized for transactional performance and integrity in mind.

The graph databases can be those which store data natively in graph format and use index-free adjacency. This means that the records physically point to each other in the database. These are highly optimized for storing and managing graphs. This is termed as native graph storage. And, the databases that form a graph overlay on top of general purpose transactional data stores or RDBMS. These do not store data in native graph format inherently. But from the user perspective, they behave like a graph database. It is worth noting the significant performance boost in using a graph data store with index-free adjacency. And then, there are graphs that compute engines which are mostly used to analyze huge amounts of data. These are mostly used in analytical use cases. Google's Pregel is the system which Google uses to build its web graph. Giraph and Pegasus are some of the distributed graph compute engines.

In graph data model, relationships are first-class citizens. This is not the case with other DBMS. We need to use foreign key dependencies or processing routines like Map-Reduce to infer connections. Graph databases provide simple abstraction of nodes and relations to build connected structures. This enables us to build sophisticated models mapping intuitively.

22.1.3 Graphs versus Other Data Models

Relational databases are initially designed to store tabular structures, something they do exceptionally. For decades, developers tried to accommodate connected, semistructured data inside relational databases. But the performance takes a hit whenever there is need to access large amounts of data to query interconnected data. Relationships exist in the RDBMS at the time of modeling, as a means to join tables. The overall structure of the data set becomes more complex and less uniform as the outlier data multiple. This leads to a lot of null-checking logic, large join tables impeding performance. This easily becomes a breaking point to evolve an existing database with dynamic business needs.

As mentioned earlier, relational databases struggle with highly connected domains. To illustrate the point, let us consider a simple and not-so-simple query in a simplified social network. The people are connected through directed lines. A directed line between Rajesh and Rupa, for example, implies that Rajesh considers Rupa as a friend but the reverse is not valid (Figure 22.2).

Let us start with a simple question, "Who are Rajesh's friends?" SQL query for this question is simple as follows:

EXAMPLE 22.1: WHO ARE RAJESH'S FRIENDS?

```
SELECT p1.name
FROM Person p1 JOIN Friends
  ON Friends.FriendID = p1.ID JOIN Person p2
  ON Friends.PersonID = p2.ID WHERE p2.name = 'Rajesh'
```

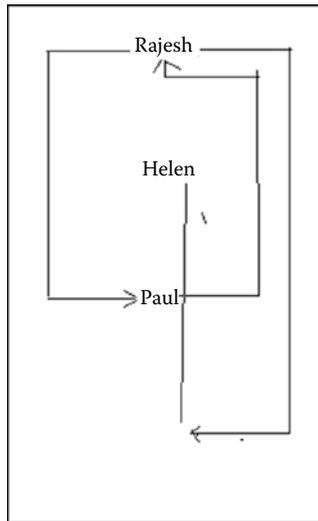


FIGURE 22.2
Simple social network graph.

Referring to the sample data, the answer is Paul and Rupa. This query is neither particularly complex nor computationally expensive as it restricts the number of columns returned and the number of rows under consideration is filtered with clause `Person.name = "Rajesh."` As we know friendship is not always a reflexive relationship, let us ask a trickier question, "Who is friends with Rajesh?"

EXAMPLE 22.2: WHO ARE FRIENDLY WITH RAJESH?

```

SELECT p1.name
FROM Person p1 JOIN Friends
ON Friends.PersonID = p1.ID JOIN Person p2
ON Friends.FriendID = p2.ID WHERE p2.name = 'Rajesh'

```

The answer to this query is Paul; Rupa does not consider Rajesh as a friend. This reciprocal query is still easy to write, but the database implementing takes more memory and time as the execution engine needs to load the whole friends table and then continue with querying who consider Rajesh as friends.

To pump up the performance, we can create an index on FriendID field of friends table, but this comes at the cost of additional processing. Things spiral down when we ask question like "Who are friends of my friends?" To get response to this question, we need to use recursive joins, which makes the query syntactically and computationally much more complex.

EXAMPLE 22.3: RAJESH'S FRIENDS-OF-FRIENDS

```

SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND FROM Friends pf1
JOIN Person p1 ON pf1.PersonID = p1.ID
JOIN Friends pf2 ON pf2.PersonID = pf1.FriendID
JOIN Person p2 ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Rajesh' AND pf2.FriendID <> p1.ID

```

Even though it deals only with friends of Rajesh's friends, executing this query is computationally expensive. As we drill deeper down into the network, things get worse and more expensive. We might retrieve three degrees of friendship in a reasonable amount of time,

but this quickly deteriorates as we move toward four, five, and six degrees of friendship due to the increasingly memory-consuming recursive joins.

Relational databases are schema oriented, whereas graph databases are occurrence oriented. Traditional business applications such as payroll, order processing, and hospital management systems conform with relational approach. In the case of graph model, the schema is not known before [17].

The connected data have performance penalties not only in relational model but also in NoSQL databases. Most NoSQL databases other than graph databases are designed to store disconnected documents or values making it difficult to use for connected data and graph.

A popular strategy for adding relationships to such disconnected stores is to embed record identifier of one record in another record. But this requires joining aggregates at the application level, which quickly becomes prohibitively expensive. As the record updates or deletes, these references are in sync with the rest of the data. Otherwise, they pollute the data with dangling references and affect the query performance. Another problem with this strategy is that the links cannot be traversed backward as there are no identifiers. The documents stored as separated documents which increase the write latency, and traversal becomes prohibitively expensive at multiple hops.

In this social network, as in so many real-world cases of connected data, the connections between entities do not exhibit uniformity across the domain—the domain is variably structured. A social network is a popular example of a densely connected, variably structured network, which resists being captured by a one-size-fits-all schema or conveniently splits across disconnected aggregates. Our simple network of friends has grown in size (there are now potential friends up to six degrees away) and expressive richness. The flexibility of the graph model has allowed us to add new nodes and new relationships without compromising the existing network or migrating data—the original data and its intent remain intact.

The graph offers a much richer picture of the network. We can see who LOVES whom (and whether that love is requited). We can see who is a COLLEAGUE_OF whom and who is the BOSS_OF them all. We can see who is off the market because they are MARRIED_TO someone else; we can even see the antisocial elements in our otherwise social network, as represented by DISLIKES relationships. With this graph at our disposal, we can now look at the performance advantages of graph databases when dealing with connected data.

Advantages of using a graph database are as follows:

- Performance in connected data over large data sets:

As the graph databases store data with index-free adjacency. Traversal of data is faster. All the nodes connected to a particular node exist in the same locality. This reduces the execution time for this recursive queries.

- Agility with incremental update of schemas. Iterative testing functionalities through APIs help the developers ship new functionalities in much lesser times.
- Easier data modeling: The schema of the data is not predefined, and it can evolve, making it easier to model data. Inherently, we use graph to model the schema even in relational data modeling. Developing graph database schema is more toward our logical data model.
- Flexibility:

Many times, the data structure and schema are not fully developed for the domains which are understood less. Flexibility of graph databases in evolving schema with the understanding of the domain greatly helps. Unlike in relational data model, attributes can be easily added or deleted.

We can easily add new nodes, new labels, and new kind of relationships without disturbing the existing structure or user queries. We do not have to design the model with exhaustive detail ahead of time. Also, this flexibility implies that we need to perform fewer migrations, reducing maintenance risk and overhead.

22.2 Use Case: Social Contacts

To understand the concepts of graph databases, we will use a simple social network. The people are connected to each other as either {Family, Friends, Knows}. The social network also contains information such as books a person likes or food items a person likes to eat. In this example, we will be storing the contact details of the person and his date of birth. We will use this to ask some interesting questions on the data.

22.3 Introduction to Neo4J Graph Database

Neo4J is a graph database management system developed by Neo technologies. It is an ACID (REFERENCE)-compliant transactional database which stores data in native graph format. It is the most popular graph database. It is available in open-source community edition and commercial edition with AGPL license [11].

Neo4j is implemented in Java and has language binding in all the popular languages such as Python and C++. Neo4j uses Cypher Query Language to query the database through API or HTTP calls. Other graph query languages are RDF query language SPARQL, GraphQL, and imperative, path-based query language Gremlin. We can also use Neo4J as a graph compute engine on top of base system of records in which any transactional data can be stored like Mysql and DB2 (Figure 22.3).

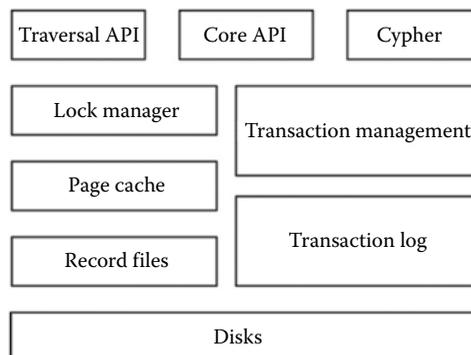


FIGURE 22.3
Neo4j high-level architecture.

22.3.1 Data Types and Data Access Mechanisms Supported

Neo4j stores data using labeled property graph model. A labeled property graph is made up of nodes, properties, relationships, and labels. Nodes can contain properties. We can think these nodes as a set of key–value pairs. In Neo4j, keys are strings and values are string, primitive data types (int, float, and boolean) and array of these types.

We can tag nodes with one or more labels. Labels help us in group nodes together and providing a context to the nodes. These indicate the roles each node plays within the whole data set. Since a node can fulfill multiple roles in a graph, we can add more than one label to a node in Neo4j. By using labels to nodes, we are declaratively indexing nodes.

Relationships connect nodes which can possibly have different labels. A relationship can always exist only in-between two nodes. The relationship is named and has directions. Relationships can also have properties. This is very useful in providing extra metadata to graph algorithms and additional semantics to relationships (like weight, etc.). These relationships are first-class citizens of graph database. These relationships form paths naturally. Every query or graph traversal involves in following these paths. Relationships can help in segregating and connecting portions of graph [18].

22.3.2 Cypher Query Language

Cypher is easy to read and understand. It defines graph manipulation intuitively using graphs. Cypher enables user to query database to match a specific pattern. The way to describe the pattern is to draw them using ASCII art [10]. [Figure 22.4](#) represents the equivalent Cypher ASCII art to describe the simple social network we have seen earlier in [Figure 22.1](#). The pattern describes four mutual friends.

This pattern describes a path that connects nodes called Paul with Rajesh who in turn connects with Rupa who is connected to Helen node. Here, the Rajesh, Rupa, Helen, and Paul are identifiers, much like variables in other languages. While describing patterns, we can use these identifiers to refer to the same node multiple times. This helps in getting around the fact that the query language gives us flexibility to query in only one dimension, where the graph itself is laid out in two or more dimensions.

Although the previous Cypher pattern is logically correct, it is not useful. We have no way to distinguish different nodes in the database as it does not refer to any particular data. This is where the labels and node properties kick in. They help us in locating relevant elements for processing ([Figure 22.5](#)).

We have given a label person to each node and assigned an identifier name to each node. Using this, we can refer to individual nodes easily. For example, the Paul identifier refers to a node in the database with a label as Person and name property set to “Paul Godman.”

```
(Paul)-[:FRIEND]-(Rajesh)-[:FRIEND]->(Rupa)-[:FRIEND]-(Helen)
```

FIGURE 22.4

Social network expressed using an ASCII diagram.

```
(paul:Person {name: 'Paul Godman'})-[:FRIEND]-
(rajesh:Person {name: 'Rajesh'})-[:FRIEND]->
(rupa:Person {name: 'Rupa'})-[:FRIEND]-(helen:Person {name: 'Helen'})
```

FIGURE 22.5

Cypher ASCII for social network with labels and properties.

You might have observed that the graph diagrams above tend to contain specific nodes and relationships. This is not the case with relational model containing classes of objects. Typically, smaller subgraphs are used to illustrate the whole graph structure. Simply, the data model is specified by example instead of schema.

Any Cypher query forms anchor points for one or more part of the pattern to specific part of the graphs using meta-information about existing indexes, constraints, property predicates, and labels. Once these anchor points are formed, then the query engine traversals the graph around the anchor points to match the pattern specified. On most occasions, these anchor points are determined automatically; however, when the data set is huge or dense, then providing additional clauses and restrictions on the query might be helpful in increasing the execution efficiency.

Akin to other query languages (SQL/GRAPHQL), Cypher is made up of clauses. One of the basic queries in Cypher consists of Match clause followed by return clause. A simple query that returns a list of friends of Rajesh from the earlier example looks as follows:

```
MATCH (a:Person {name:'Rajesh'})-[:FRIEND]->(b)
RETURN b.name
```

Let us have a more detailed look of above query.

22.3.3 Match

The Match clauses form most of the Cypher queries we fire. We represent the nodes and relationships using ASCII characters. Nodes are drawn using parentheses “()”. The property predicates of node and relationships are specified inside curly brackets “{}” as key-value pairs.

The relationships are represented as double dashes with a greater than or lesser than signs. The signs indicate the direction of the relationships. We can also represent bidirectional relationships with just using double dashes “—”. Given that the relationships can be named, we select the relationship using the name of relationship prefix with colon “[:RELATIONSHIP_TYPE]”. Node labels can be specified with similar prefix with colon “(:NODE_LABEL)”.

In the above query, we are seeking a node labeled with Person and the name property of the node is “Rajesh” which is bound to identifier a. Thereafter, this identifier refers to the node representing Rajesh throughout query execution.

This start node is part of a simple pattern (a)-[:FRIEND]->(b). This pattern describes a path that contains two nodes. One of which is bound by identifier a and the others by b. These nodes are related to each other by a single friend relationship.

In theory, the above pattern can occur many times throughout the data set. For anchoring parts of the above query to some location(s) in graph we look nodes labeled “Person,” with name property value as “Rajesh.” This bounds us to a specific node in the graph. Cypher matches the rest of the pattern to graph neighboring this anchor point. It discovers nodes which can be bound to other identifier. Identifier a is always anchored to the Rajesh node, and b can be bound to a sequence of nodes.

Alternatively, the predicate anchoring can be specified in the WHERE. The following query gives the same output as earlier:

```
MATCH (a:Person)-[:FRIEND]->(b) WHERE a.name = 'Rajesh'
RETURN b.name
```

22.3.3.1 RETURN

This clause specifies what needs to be returned to the user. In SQL parallel, this is like selecting the columns. In the above example, we are interested in knowing the names of Rajesh's friends, that is, property of nodes bound by `b`.

22.3.4 Other Cypher Clauses

The other clauses we can use in a Cypher query include the following:

WHERE to provide filter criteria on matching patterns.

CREATE and CREATE UNIQUE: Used to create nodes, constraints, indexes, and relationships.

MERGE: This clause ensures that the supplied pattern exists in the database either by reusing existing nodes and relationships or by creating new ones.

DELETE: Deletes the nodes, relationships, and properties that match with the pattern. To delete the relationships attached to a node along with the node, you can use **DELETE DETACH**.

UNION: Merges results from two or more queries.

SET: Used for updating node/relationship properties for matching supplied patterns.

FOREACH: Used for performing an updating action for each element in a list.

22.3.5 Relational Data Model for Social Contacts Application

As with any other data modeling, the first step is to agree on entities in the domain [12]. We will then move on to the interrelation between entities, how and when they are transformed. This step is done informally using mind maps or using similar techniques. We arrive at a rough picture about the conceptual data model.

Once we have a rough idea about the domain and the data model, we formally capture this agreement on more rigorous form such as entity–relationship (ER) diagram. This transformation provides us with a chance to refine our domain vocabulary. [Figure 22.6](#) shows the ER model for our social network application. Once finalized, we will use this ER model to convert to physical database model.

We map this logical model into tables and relationships. We normalize the tables to eliminate data redundancy. Often, this can be as simple as rendering the ER diagram into a tabular format. Yet, while converting this into a relational model, we added a great deal of complexity in the form of foreign key relationships to support multitable joins at query time. Yet, this clutters the domain data with additional metadata. Metadata serves the purpose of database not the user ([Figure 22.7](#)).

Even though we arrived at a relatively faithful representation of the domain, our design work is not complete. Even though this model is substantially complex, it has no duplicate data. Yet, this model is not fast enough for real-world usage. In theory, this model is sufficient for ad hoc queries. But in practice, this model needs optimizations to special access patterns by creating indexes. We need to fit in our logical data model to suit the database engine. Denormalization is the name given for this.

Denormalization necessitates duplicating data to gain query performance. This is primarily done to reduce the number of joins and associated performance penalties. For example, a person might have multiple contact numbers. In a fully normalized model, we

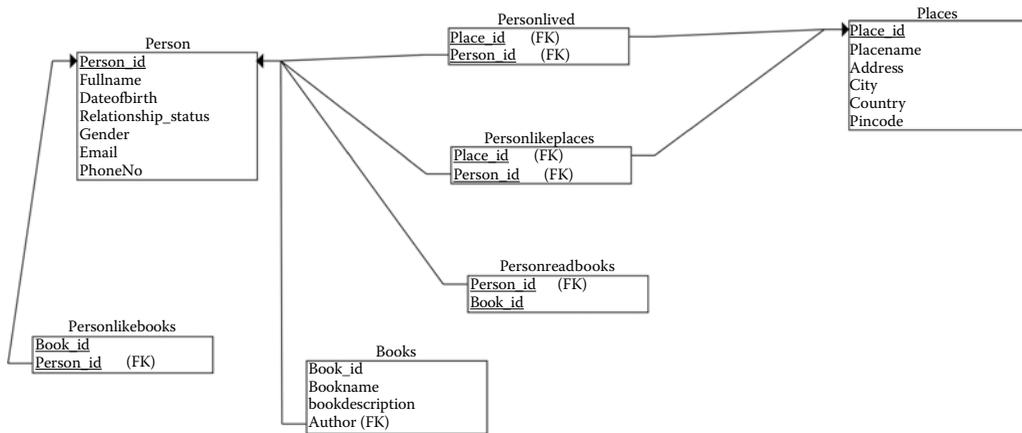


FIGURE 22.6
ER model for social contacts application.

might have to store these in separate CONTACTS table. Often, it is a common practice to add one or more columns to store persons' contacts.

Although denormalization is safe thing to do, it is not a trivial task. One needs to have expertise to align the denormalized model with the RDBMS and storage tier. It is tempting to think that the effort we put in normalize–denormalize design is acceptable. As it is a one-time thing, the effort is amortized over the lifetime of application. This is an alluring notion but does not reflect the reality. The reason for this notion is the assumption that the data model is stable.

The system and our understanding of the domain change throughout the lifetime of application. During the design and development of the project, the data models undergo substantial revisions. Once the data model is imported into production, it is almost impossible to modify the model. These changes might be due to new features or new statutory requirements. The technical process of introducing these structural changes is called migrations. Data migrations require a systematic approach. Unlike code refactoring, database migrations often take weeks or even months. The whole process of migration is risky, expensive, and slow. The data might be inconsistent or copied partially if we are not careful.

22.3.6 Graph Modeling for Social Contacts Application

In the above section, we have seen how the basic conceptual model beats down into applications' storage model. The rigid schemas do not support rapid changes. Agility, a core requirement for all modern applications, is not met. We will look at the graph model for this application. Using this model, we will not compromise on the application performance. At the same time, we make the physical model more accessible to business users, helping with rapid evolution of data in tandem with business needs [13].

In the earlier stages of modeling, we concur upon our initial domain understanding. We follow similar approach to relational modeling using mind maps and whiteboard sketches. Instead of converting the whiteboard sketches into tabular format, we enrich the graph structure. We ensure that every entity has attributes as properties, relevant roles as labels, and their connections to neighboring entities as relationships.

```

create table Person(
  person_id int not null auto_increment,
  fullname varchar(50)not null,
  gender char(1), email varchar(254),
  phoneno varchar(15),
  relationshipstatus char(2),
  date of birth date, primary key (person_id)
);

create table Book(
  book_id int not null auto_increment,
  bookname varchar(50) not null,
  book description varchar(100),
  author int not null, primary key (book_id),
  foreign key (author) references Person (person_id)
);

create table Place(
  place_id int not null auto_increment,
  place name varchar(100) not null,
  address varchar(100),
  city varchar (50),country varchar(50),
  pincode varchar(10), primary key(place_id)
);

create table Person_Books (
  person_id int not null,
  book_id int not null,
  primary key (person_id, book_id),
  foreign key (person_id) references Person(person_id),
  foreign key (book_id) references Book(book_id)
);

create table Person_livedPlaces (
  person_id int not null,
  place_idv int not null,
  fromdate date,
  todate date default '99991209',
  primary key (person_id, place_id),
  foreign key (person_id) references Person(person_id),
  foreign key (place_id) references Place(place_id)
);

createtablePerson_likePlaces(

```

FIGURE 22.7
Relational data model.

Domain modeling is not about having a context-free view of reality. Instead, it is about drawing intentional abstractions to fit the application requirements. Graph model is isomorphic to the domain model we have developed. Just by ensuring domain correctness, the graph model is greatly enriched.

Now, we improve the whiteboard sketch by placing appropriate role-specific labels and properties. By doing this, each node can fulfill its role-specific responsibilities. For example, a node can be labeled *Person* referring to a user, and another can be labeled as *Book* to refer to a book. We also need to see that every node is placed in correct semantic context. As in, person can be writing a book or he might like a particular book. We do this by creating named and directed relationships between different nodes. These relationships can also have attributes, for example, an attribute which say how much I like a Movie.

Sometimes, a node can have multiple labels either referring to different roles it may play in the context or to show the generalized class one node might belong to. The node labeled author is also a node referred to a person who might also be a user.

Essentially, that is all we need to do. We have no need to do any normalization and to optimize the structure according to RDBMS. We are all set to move into database which is almost a trivial task.

22.3.7 Testing the Model

We have polished our model. The next step is to test if this model can answer our queries. As we have noted earlier, the graph is adaptive and evolves with our ever-evolving business needs. However, the early design decisions if taken erroneously can hamper application performance or adaptability. So, by careful review of the domain model, we can avert these drawbacks.

We can use two techniques just for this cause. The first and the simplest one is to just check if the graph reads well as we traverse it. In this technique, we start at a node and then follow the relationships to other nodes. If we are able to create sensible sentences, then the model is consistent. For example, from our social contacts application data model, we can create following sentences: “Helen like book Nirmal written by Sakshi who know Robin her favorite author” or “Robin has written the monk who sold his Ferrari.”

Furthermore, we need to confirm that graph model supports the real-world queries we wish to run on it. This might require understanding of how users going to interact with the application. Following are few ways a user has in our social contacts application. A user might ask how to get introduced to his favorite author. We need to identify another user in his social circle with a connection to author or find a place to stay in the place he is visiting. For this, we need a user who lives in the same place the querying user visiting. We use the following queries for the same effect:

```
MATCH(user:Person)-[Friend|Family]-(mutuals:Person)-
[*1..2:Friend|Family]-(author:Person)-[:Author]->(book:Book),
(book:Book)<-[:Likes]-(user:Person) WHERE user.name = 'Rajesh' Return
distinct mutuals.
```

The Match clause describes variable length path between author and Rajesh. The arrow tip indicates the direction of relationships. The name in the square brackets is the name of relationship between nodes. Starting from the user who wishes an introduction, the traversal processes. We have to find a path through friends and family of the user who are in turn friends or family with the author. “Return distinct” query keyword ensures that unique mutual contacts are returned in results. This query is readily supported by our graph model.

We can understand more about the users by integrating multiple domains. Generally, we store domain-specific data in silos. It is hard for us to cross-refer different data silos. And, due to this, the network effects between the silos are lost. Business insights that might have been easily understood by integrating these different silos are lost. For example, you have a firm with multiple divisions, and there is one common customer. Since we are storing data in silos, we do not have an insight that we can provide better customer service. Another might be that we have an e-commerce store that stores the transactions and orders. And, we also build a social network like our use case, where we are storing user likes and dislikes. If we integrate both these domains, we can have much richer understanding of customer and even use the social network to increase the sales by offering relevant deals.

To build this insight, we need to integrate domains without sacrificing individual domain details. We can use the labeled property graph model with different labels set to differentiate different subgraphs/domains.

22.3.8 Delete Command

```
Match (n:Person {fullname: 'Siddhu'}) Delete n
```

22.3.9 Loading Large Amounts of Data from CSV

```
USING PERIODIC COMMIT
```

```
LOAD CSV WITH HEADERS FROM "http://portal.graphgist.org" AS line
```

22.4 Creating Social Contacts Graph

To create the Social Contacts graph, we use CREATE statement to build the overall structure of data. This Create statement is processed in a single transaction by the Cypher runtime. This assures that if the statement is successful, the entire graph is present in the database. If transaction fails, no part of the graph will be present in database.

```
CREATE (:Artist { name: line.Name, year: toInt(line.Year)})
```

22.4.1 Load Books from CSV

```
LOAD CSV WITH HEADERS FROM 'http://neo4jhandson.s3-website-ap-southeast-1.amazonaws.com/Book.csv' AS line
```

22.4.1.1 Create Books

```
CREATE (:Book { book_id: line.book_id , bookname: line.bookname,
bookdescription: line.bookdescription , author: line.author})
```

```
CREATE (:Book {book_id:1, bookname:"Joy",bookdescription:"Some book on
Joy",author:1}), (:Book {book_id:2, bookname:"Nirmal", bookdescription:"The
purest of emotions", author:5}), (:Book{book_id:3,bookname:"Monk who sold
his ferarri",bookdescription:"Insirational story on life", author:7}),
(:Place {place_id: 1, placename: "Sanpada", address: "Dream Homes", city:
"Navi Mumbai", country:"India", pincode:"400705"}), (:Place {place_id: 2,
placename: "GT Road", address: "Savahana", city: "Chennai",
country:"India", pincode:"500200"}), (:Place {place_id: 3, placename:
"Paris", address: "La arcade", city: "Paris", country:"France",
pincode:"131421"}), (:Place {place_id: 4, placename: "Bay Area", address:
"Multiplex", city: "San Fransico", country:"USA", pincode:"131563"})
```

22.4.1.2 Book Author Relationship

```
MATCH (person:Person), (book:Book)
MERGE (person)<-[r:AUTHOR]-(book) WHERE book.author = person.person_id
RETURN person.fullname, book.bookname
```

22.4.1.3 Book Like Relationship

```
MATCH (person:Person {person_id: 1}), (book:Book {book_id: 1})
      MERGE (person)-[r:LIKES]->(book) RETURN person.fullname, book.
      bookname
```

22.4.1.4 Person Likes Place

```
MATCH (person:Person {person_id: 4}), (place:Place {place_id: 2})
      MERGE (person)-[r:LIKES]->(place) RETURN person.fullname, place.
      placename
```

22.4.1.5 Person Lives Place

```
MATCH (person:Person {person_id: 4}), (place:Place {place_id: 2})
      MERGE (person)-[r:LIVES {fromdate: '20100202', todate:
      '99991209'}]->(place)
      RETURN person.fullname, place.placename
```

While the first Cypher code creates labeled nodes with respective properties, the later codes connect these nodes with relationships (with properties as necessary). The identifiers we have used are available as long as the current query scope. If we require these easy node accesses, then we simply need to create index for a particular label and key property combination.

The above commands will not introduce any accidental complexity into graph unlike in relational data model. The domain meta-model, that is, the structure of domain data and the relationships, is segregated from business data which is wholesomely embedded into node/relationship properties. We need not worry about cardinality constraints and foreign key constrains that might pollute the data. These interconnections are explicit in the form of semantically rich relationships.

In the second set of queries, we are modifying the graph. We can always use CREATE statement to simply add to the graph. We have used MERGE command as it semantically ensures that the structure of nodes and relationships which may or may be present in the graph is in order. As a thumb rule, we use CREATE to add nodes and MERGE for modifications which does not allow duplication.

```
USING PERIODIC COMMIT
```

```
LOAD CSV WITH HEADERS FROM 'http://neo4j.com/docs/3.0.0-M05/csv/artists-
with-headers.csv' AS line
```

In case you want to move data in bulk from a CSV file, you can use the above command. Also, it is advised to periodically commit the data mainly if you are moving large chunks of data so that the data can be flushed to long-term memory.

22.4.2 Querying the Database

Now that we have created the database, let us query it using Cypher. In Cypher, we always start queries from one or more known starting points in the graph. For example, a person name “Rajesh” and a book name “Airavali.” These starting points are called bounding nodes. These help the query engine to localize the query in the graph. Cypher

runtime uses any labels, property predicates supplied in MATCH and WHERE clauses in conjunction with the indexes, constrains to find the starting points that anchor our graph patterns.

For instance, if we want to find out the details about the books a person, say Rajesh, likes, we had to start with Rajesh node by specifying the Person label and name property. Whereas if we want to find out the most popular tourist place, we will start with the Place label and count the number of people like that place. This information about the property pattern we specify binds the query to specific part of the graph, giving us the starting points from which we need to match patterns in immediately surrounding nodes and relationships.

22.4.3 Indexes and Constrains

Indexes help optimize the process of finding specific nodes. This is one of the features graph databases have similar to the relational databases. We need to pick certain nodes directly sometimes, like when we need to find starting nodes for traversal by combination of labels and property values.

For this purpose of efficient node lookup, Cypher allows us to create indexes per label and its property combinations. We can also specify constraints that assure uniqueness. In our social contacts application, we need to look up a Person directly. So let's create a unique index on the Person label:

```
CREATE INDEX ON :Person(name)
```

To ensure that all country names are unique, we can add a uniqueness constraint:

```
CREATE CONSTRAINT ON (c:Person) ASSERT c.name IS UNIQUE
```

22.4.4 Finding Information Patterns

We use MATCH clause in Cypher to match the patterns. We use the same syntax used in CREATE clause using ASCII art to describe the state of graph to discover patterns using MATCH clause. We have already seen simpler Match query. Let us look at a more complex pattern to all Persons Living in New York who likes to stay in New York.

```
MATCH (city:Location {name:'New York'}),
      (person:Person)<-[:LIVES]-(city) <-[:LIKES]-(person)
      RETURN DISTINCT person.name AS names,
```

We can constrain graph matches using WHERE clause. Using WHERE clause permits us to eliminate subgraphs by either eliminating certain paths by stipulating certain path that needs to be presented, nodes with certain names or elements or matched nodes, and their property values.

Cypher's RETURN clause allows us to perform some processing on the matched graph data before returning it to the user that executed the query.

```
RETURN DISTINCT play.title AS play
```

DISTINCT keyword ensures that we return unique results. Because by traversing different paths we can get to the same person, we use DISTINCT filters to remove these

duplicates. We can enrich the output by aggregating, ordering, and filtering returned data, for example, the number of people living in a particular city by using the following command:

```
RETURN count(names)
```

If we want to rank the city by the number of people visiting, we will need first to bind the VISITED_BY relationship in the MATCH clause to an identifier, called *p*, which we can then count and order.

Using WITH clause, we can chain together several matches something similar to piping query results from one query to another. It is not always practical to write all the constraints or clauses in a single MATCH clause.

In the following example, we find the people visited New York and order them based on the year in which they visited, latest first. Using WITH, we then pipe the results to the RETURN clause, which uses the collect function to produce a comma-delimited list of person:

```
MATCH (person:Person)-[w:VISITED]->(city:Location {name: "New York"})
  WITH person ORDER BY w.year DESC
  RETURN collect(person.name) AS people
```

By executing the above query against our graph, the output will be as follows: {"Siddhu", "Alekhya"}.

We develop the data model feature by feature and user story by user story. This will ensure that we identify the relationships our application will use to query the graph. A data model developed in line with iterative and incremental delivery of application features will look quite different from data-model first approach. However, the model will correct representation motivated by the demands of application features.

Graph databases provide for the smooth evolution of our data model. Migrations and denormalization are rarely an issue. New facts and new compositions become new nodes and relationships, while optimizing for performance-critical access patterns typically involves introducing a direct relationship between two nodes that would otherwise be connected only by way of intermediaries. Unlike the optimization strategies we employ in the relational world, which typically involve denormalizing and thereby compromising a high-fidelity model, this is not an either/or issue: either the detailed, highly normalized structure or the high-performance compromise. With the graph we retain the original high-fidelity graph structure, while at the same time enriching it with new elements that cater to new needs.

22.5 Common Use Cases

In this section, we will look at some of the most common ways graph databases are used across industries and research [4,5]. We will briefly describe what graph characteristics commonly applicable to these test cases to generate unusual business advantage. We will dwell into these use cases noting that graph databases might not be the best solutions for updating sets of data like complex financial transactions [6].

22.5.1 Social Network

As the name itself says, the social network is a network of people, their likes, dislikes, and memories they share with the world. We already use Facebook to get connected with our friends and family [16], LinkedIn to keep in touch with our professional network, Twitter for real-time conversations and discovery, and networks like Quora and Medium to quench our intellectual thirst. By the help of social applications, organizations can gain competitive and operational advantage by leveraging the connections the users make on their platform. By combining discrete information about the users and their relationships, organizations can remove the boundaries between people and even predict user behavior.

Graph data models are a natural fit for this openly relationship-entered domain. Social networks by virtue of direct or indirect relationships allow users to rate, review, and even discover other users and things they care about. By understanding who interacts with whom and how people are interconnected, we get immense insight into the unseen forces influencing individual behaviors.

The direct relationships occur when a user directly likes his friends' photo or stating directly that he had worked with a particular colleague on some project. Indirect relationships occur when the relationships occur by way of an intermediary. For example on Facebook, the application suggests people you may know by analyzing mutual friends, location, and places you worked or studied [2].

22.5.2 Recommendation Engines

By application of inferential and suggestive capabilities, recommendation engine predict and suggest things users are interested in. On the flipside transactional business, applications use deductive or sequential logics to create user values.

Relationships are established between people and other people, products, and services whatever is relevant to the recommendations that are employed. The relationships are established based on the user behavior as they purchase, consume, or in any way interact with the applications. The recommendation engine can utilize and then identify things of interest for a particular user or group, or identify users or group who will be interested in a particular thing. In the first case, the base point is the users or group characteristics which are correlated with people of similar characteristics, and in the latter case, the characteristics of resources are correlated with similar products and the users of those products.

Making an effective recommendation involves understanding not just the relationships but also the quality and strength of those connections, which are easily expressed using a property graph. Taken together, recommendation engines and social networks provide key differentiator in the areas of retail, sentiment analysis, search [9], and knowledge management. Storing and querying these data in a graph databases enable applications to provide dynamic real-time recommendations instead of precalculated static results.

22.5.3 Geospatial Data

Geospatial is the native graph use case. Leonhard Euler wrote the first paper in graph theory on Seven Bridges of Königsberg in 1736 [3]. Geospatial applications range from finding points of interest in a bounded area to calculating efficient routes between locations in an abstract network such as air, road, and rail network for calculating intersection of two or more regions.

Due to the schema-free characteristic of graph databases, geospatial data can reside alongside other kinds of data—delivery orders for example. This allows us to ask complex

multidimensional queries. This fact can be leveraged by logistics technology firms to route and dispatch deliveries efficiently and in time. We can even further ask questions like how to get to one-from-another swiftly. E-Bay uses graph technology to maintain its delivery logistics [1].

Graph databases' geospatial applications are particularly relevant in the areas of travel planning, logistics, and telecommunications.

22.5.4 Graph Search

Using graph to provide contextual search is another interesting use of graph databases. Knowledge Graph [15] from Google is one such graph-based contextual search engine. The reason the contextual search works well with graphs is that the graphs are ideal semantic data stores. For example, graph databases are commonly implemented using triple-store concept of object, relationship, and predicate, a core concept in semantic analysis [8,9].

22.5.5 Fraud Analytics

Billions of dollars are lost by bank and insurance companies due to fraud. Traditional techniques based on pattern matching have been successful in reducing the intensity of these losses. However, traditional techniques consider individual transactions or events as discrete and try to match patterns. Significant improvement can be achieved if we look beyond the connections between these fraudulent transactions. Often, these connections go unnoticed till it is too late.

This loop hole has helped build sophisticated techniques to play the system, both by joining hands with other fraudsters and by creating illegitimately legitimate false identities. As these fraud rings grow in size, the traditional techniques take higher magnitudes of time, even if the flagging is inaccurate. Complementing the traditional techniques, if sophisticated techniques which can easily give the interconnections in real time are used, then the fraudulent losses can be further reduced.

This does not mean that we have to gather new data. We can derive significant insights from existing data just by reframing our view in terms of connected data, a graph. For active prevention of any kind of fraud, we need to be clear on two things: real-time fraud detection to stop the criminals before they do too much damage and the value of connected analysis which helps in exposing the sophisticated techniques the criminals use to beat the system [14].

Let us have a look at how common stolen credit card fraud groups function and how graph can help in exposing these frauds. To fight back crime and to be against criminals, it is important to understand how they operate. We use credit cards to pay for all sorts of stuff, our departmental purchases, food, medicine, flight tickets, online purchases, and even dining. Now, if someone gets access to your credit card details, the fraudster can empty the account the card is attached to.

Let us say that we are working with a credit card company to flag the suspicious places. We ideally want to catch the criminal as quickly as possible and disabling him to make any more purchases. So how do we do this? The data we have are the customer transactions at different merchants and transactions which are flagged as disputed by customers. The data we have are in tabular format as shown in [Figure 22.9](#) [7].

In [Figure 22.8](#), we have a peek view of the credit card transactions at different merchants. Transactions marked in dark gray are disputed transactions.

Looking at the data in [Figure 22.9](#), are you able to recognize any pattern from the list? When looking at discrete records, it is really difficult to understand the relationship

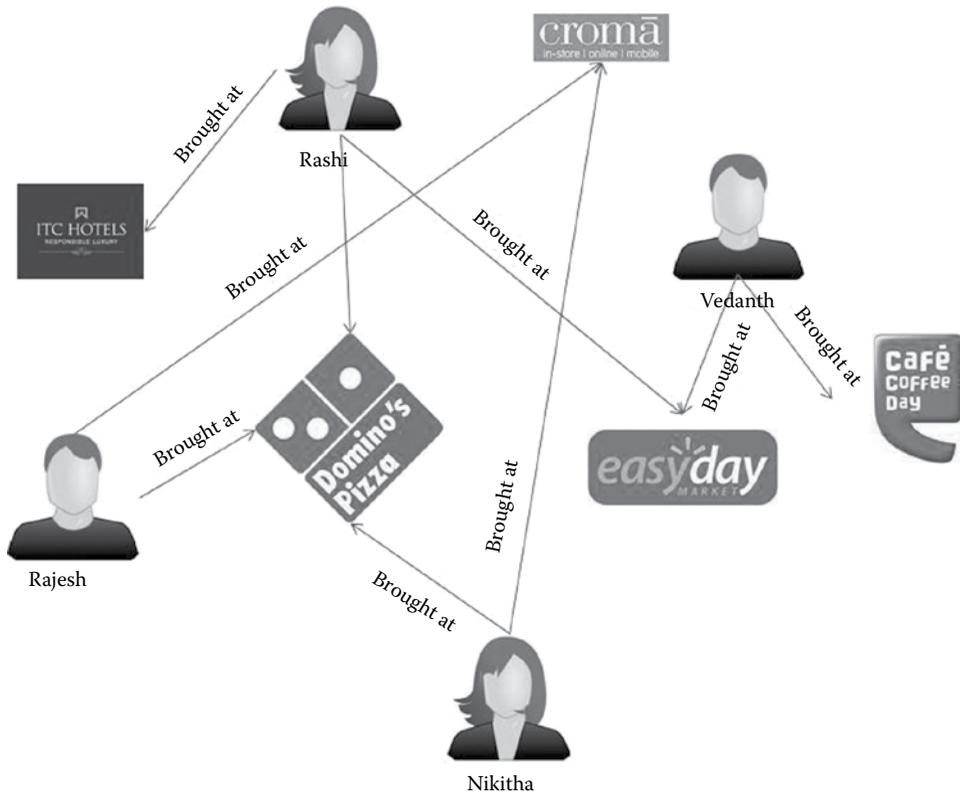


FIGURE 22.8
Credit card transactions.

Transaction Date	Store Name	Customer Name	Amount(INR)	Status
20/08/2015	Easy Day	Rashi	1800.02	Disputed
20/08/2015	Easy Day	Rashi	1500.53	Disputed
20/08/2015	Easy Day	Rashi	3000.00	Disputed
12/10/2015	Croma	Vedanth	2500.00	Disputed
12/10/2015	Croma	Vedanth	1400.00	Disputed
12/10/2015	Croma	Vedanth	3500.00	Disputed
13/11/2015	Croma	Vedanth	2230.00	Disputed
08/06/2015	Shopper Stop	Rajesh	2000.00	Disputed
08/06/2015	Shopper Stop	Rajesh	1402.00	Disputed
08/06/2015	Shopper Stop	Rajesh	1221.00	Disputed
08/06/2015	Crosswords	Rajesh	230.00	Successfull
12/03/2015	Inox	Nikitha	400	Disputed
12/03/2015	Inox	Nikitha	500	Disputed
12/03/2015	Inox	Nikitha	200	Disputed

FIGURE 22.9
Customer transactions flagged as disputed.

between these records. Instead, let us try to use graph visualization to understand the links between data points. To get the above list of fraudulent transactions we can issue the following Cypher query:

```
MATCH (victim:person)-[r:HAS_BOUGHT_AT]->(merchant)
WHERE r.status = "Disputed"
RETURN victim.name as customer_name, merchant.name as store_name,
r.amount as amount, r.time as transaction_time
ORDER BY transaction_time DESC
```

Victim is a person who has flagged a transaction from a merchant as disputed. The above query gives us the list of customers and merchants involved in the disputed transactions. To find the actual culprit, we use the transaction date of the first reported fraudulent transaction. So, to find those transactions, let us fire the following query:

```
MATCH (victim:person)-[r:HAS_BOUGHT_AT]->(merchant)
WHERE r.status = "Disputed"
MATCH victim-[t:HAS_BOUGHT_AT]->(othermerchants)
WHERE t.status = "Undisputed" AND t.time < r.time
WITH victim, othermerchants, t ORDER BY t.time DESC
RETURN victim.name as customer_name, othermerchants.name as store_
name, t.amount as amount, t.time as transaction_time
ORDER BY transaction_time DESC
```

Now, our job is to simply find the common merchants where the card is used before illegitimate/disputed transactions. We just need to tweak the above query so that we count the common merchant and the number of transactions occurred for each disputing customer:

```
MATCH (victim:person)-[r:HAS_BOUGHT_AT]->(merchant)
WHERE r.status = "Disputed"
MATCH victim-[t:HAS_BOUGHT_AT]->(othermerchants)
WHERE t.status = "Undisputed" AND t.time < r.time
WITH victim, othermerchants, t ORDER BY t.time DESC
RETURN DISTINCT othermerchants.name as suspicious_store, count(DISTINCT
t) as count, collect(DISTINCT victim.name) as victims
ORDER BY count DESC
```

The graphical representation of the response is shown in [Figure 22.10](#). The Easyday shop has been linked with the largest number of victims. From this, we can deduce that something fishy is going on at Easyday store.

Every time a fraudulent transaction has occurred, earlier to this, the credit card user has visited particular Easyday store. Using this information, the authorities and the merchant can take appropriate action. This is just one of the scenarios where the quick response is required to minimize the losses. Graph-based analytics can also be used to actively prevent fraudulent transactions. Another interesting use case for graphs is datacenter environment monitoring. The complex interdependencies and dynamic nature of IT landscape make a case for modeling graph to monitor the infrastructure. Using a graphical representation, we can easily pin point the problem in application delivery to a user, whether it is a security event or a network failure event or even failure of particular app. If we can traverse through the graph by the definite node, we can arrive at the root cause of the disruption. The real-time events in the datacenter are auto logged into the database

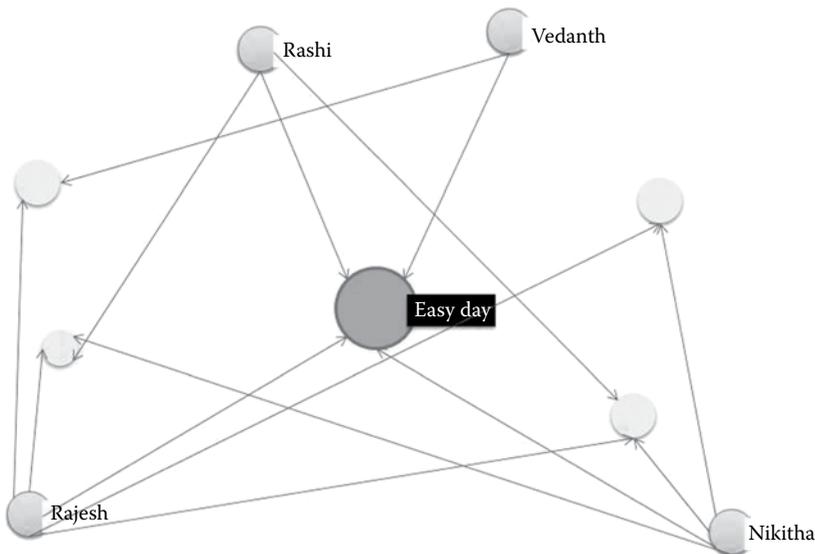


FIGURE 22.10
Fraudulent transactions response.

using a complex event processing techniques. More often than not, the problems that arise in datacenters involve a complex set of software, hardware, and human interdependencies, in which manual monitoring can quickly become challenging as the network size increases. Given the highly dynamism and higher level of interdependencies, relational data model is not an easy fit for datacenter monitoring. Some other interesting use cases of graph databases are authorization and access control, investment management, and finance. Firms like Ravel Law are using graph tools and analytics to improve search for legal profession. Another interesting use case is visualizing and analyzing protein interaction networks, which displays the clustering of proteins or patterns of interactions. The understanding of these clusters and patterns is essential to comprehend metabolic processes. Some other social graph-related applications are social media marketing and churn analysis.

Whether it is personalized product recommendations or maintaining network and IT infrastructures, or web apps adding social features or enterprises redefining identity and access management, organizations are looking at graph data models to store and query data.

Appendix

Installing Neo4j:

I'm Using AWS to run the tests and tutorials.

Create EC2 instance; Create security groups such that you can access from terminal.

Connect to Ec2 Install

Sudo yum update -y to update local repositories

I'm using Docker to separate the container mysql and neo4j

Install Docker using Amazon repository using the following command.

```
sudo yum install -y docker
```

start the docker service

```
sudo service docker start
```

Change user group to attach user group to docker

```
sudo usermod -a -G docker ec2-user
```

```
docker info
```

to pull Neo4J image in docker

```
docker pull Neo4j
```

Run the docker container mapping 7474 to 80 ec2 port so that we can access this from outside

```
docker run --detach --name neo4j -e AUTHOR="Sid" -p 80:7474 -p 443:7473 -v=$HOME/neo4j/data:/data neo4j
```

Once we installed the Neo4j Installation, we can check if everything is working fine just by going to a web browser and entering the public IP address of EC2 instance running. You should arrive at a Neo4j demo page.

For AWS EC2 tutorial: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

For Docker Tutorial: <https://docs.docker.com>

For interfacing Neo4j with Python: <https://marcobonzanini.com/2015/04/06/getting-started-with-neo4j-and-python/>

References

1. Babu, R. V. and Venkata Rao, M. Virtual market using logistics optimization. *International Journal of Advanced Networking & Applications* 1(5), 307–313, 2010.
2. Christakis, N. A. and Fowler, J. H. *Connected: the Amazing Power of Social Networks and How They Shape Our Lives*. London: Harper Press, 2011.
3. Euler, L. *The Seven Bridges of Konigsberg*. Wm. Benton, 1956.
4. GraphGist.com
5. <http://neo4j.com/graphgist/3ad4cb2e3187ab21416b/>
6. <https://dzone.com/articles/amazingly-cool-graph-db-use>
7. Jean. Stolen Credit Cards and Fraud Detection with Neo4j—Linkurious—Understand the Connections in Your Data. Linkurious Understand the Connections in Your Data. Linkurious, 2014. Web. March 12, 2016.
8. Jiang, H., Wang, H., Philip, S. Y., and Zhou, S. Gstring: A novel approach for efficient search in graph databases. *IEEE 23rd International Conference on Data Engineering* (pp. 566–575), Istanbul, Turkey. IEEE, 2007.
9. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases* (pp. 505–516), Trento, Italy, October 4–6. VLDB Endowment, 2005.

10. Kaur, K. and Rani, R. Modeling and querying data in NoSQL databases. In *IEEE International Conference on Big Data* (pp. 1–7), Hyatt Regency Santa Clara, California, October 6–9. IEEE, 2013.
11. Neo4J—Graph database Web. January 2, 2016.
12. Nijssen, G. M. and Halpin, T. A. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. Prentice-Hall, Inc., New Jersey, USA, 1989.
13. Robinson, I., Webber, J., and Eifrem, E. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Inc., 2015.
14. Sadowski, G. and Rathle, P. Fraud Detection: Discovering Connections with Graph Databases. White Paper-Neo Technology-Graphs Are Everywhere, 2014.
15. Singhal, A. Introducing the Knowledge Graph: Things, not Strings. *Official Google Blog*, 2012. <https://googleblog.blogspot.in/2012/05/introducing-knowledge-graph-things-not.html>
16. Ugander, J. et al. The Anatomy of the Facebook Social Graph. arXiv preprint arXiv:1111.4503, 2011.
17. Vicknair, C. et al. A comparison of a graph database and a relational database: A data provenance perspective. In *ACM SE 10 ACM Southeast Regional Conference*, Oxford, Mississippi, April 15–17, ACM, 2010.
18. Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., and Partner, J. Neo4j in Action. *Manning*, 81–110, 2015.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

23

Tutorial on MongoDB

Prashanta Kumar Das

CONTENTS

23.1 Why MongoDB?	401
23.2 Downloading and Installing MongoDB.....	401
23.3 MongoDB Terminologies	402
23.4 Commands.....	402
Reference	403

23.1 Why MongoDB?

Organizations of all sizes are using MongoDB. Customers number over 1000 and include the following (mongodb, 2016):

1. Cisco
2. EA
3. eBay
4. Ericsson
5. Forbes, Intuit, LexisNexis, SAP, and Telefonica

Among the Fortune 500 and Global 500, MongoDB already serves

6. 10 of the top financial services institutions
7. 10 of the top electronics companies
8. 10 of the top media and entertainment companies
9. 10 of the top retailers
10. 10 of the top telcos
11. 8 of the top tech companies
12. 6 of the top healthcare companies

23.2 Downloading and Installing MongoDB

1. Download MongoDB from URL: <http://www.mongodb.org/dr//fastdl.mongodb.org/win32/mongodb-win32-i386-2.6.4-signed.msi/download>

2. Install MongoDB.
3. Open **Window 1**, and run database process by executing **mongod.exe** from *bin* directory.
4. Open **Window 2**, and run the **database Java Script shell** by executing the **mongo.exe**. This command will connect to test database.

23.3 MongoDB Terminologies

MongoDB versus other DBMS	MongoDB	MySQL	Oracle	Informix	DB2
Database server	mongod	mysqld	oracle	IDS	DB2 Server
Database client	mongo	mysql	sqlplus	DB-Access	DB2 Client

SQL Terms/Concepts	MongoDB Terms/Concepts
Database	Database
Table	Collection
Row	Document or BSON document
Column	Field
Index	Index
table joins	Embedded documents and linking
Primary key Specify any unique column or column combination as primary key	Primary key In MongoDB, the primary key is automatically set to the <code>_id</code> field. By default 12 digits

23.4 Commands

Help to see all the mongoDB commands

To shows all the Database

```
show dbs
```

Creating a Database

```
use student
```

In mongoDB, **use command can be executed without creating a database**

since mongoDB does not create a database until and unless some records are stored and in it.

To see the active database:

```
db
```

Creating a collection [table]:

```
db.course.count() to check whether the collection exists
```

Connecting to a Database

```
$ mongo student
```

```
Creating a Record[Document]
    db.std.insert ({name: 'John'};
    emp2={age:34};
Saving a Record
    db.employee.save(emp);
    db.employee.save(emp2);
    db.employee.find();
Updating Record
    db.employee.update({name:'John'},n {$set : {age : 27}});
    db.employee.update({name:'John'},n {$unset : {age : 1}}); //to
    delete the age field
Deleting a Record
    db.employee.remove({name='John'});
    db.employee.remove() delete all
Delete the database
    db.employee.drop
```

Reference

mongoDB. 2016. Retrieved June 11, 2016, from <http://www.mongodb.com/customers>



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

24

Introduction to Oracle NoSQL Database

Suresh Mohan and Hari Seetha

CONTENTS

24.1 Introduction	405
24.2 Architecture	405
24.2.1 Read and Write Operations	406
24.2.2 Key-Value Store	406
24.3 KVLite	406
24.4 Installation	407
24.5 Getting Started	407
24.6 Tables	408
24.7 Java Driver	410
24.7.1 Java KV Access	410
24.7.2 Java Table Access	410
24.8 Loading Data from CSV	412
References	413

24.1 Introduction

Oracle NoSQL database is a distributed NoSQL database that offers reliable and highly scalable data management. It provides multi-terabyte storage with scalable throughput and performance. It provides high availability through replicas, and it boasts of no single point failure of nodes. It uses table as data model to store and retrieve data (Alam et al., 2013). Optionally, key-value pairs and JSON documents could also be used as data models. The data consistency and latency are clearly predictable based on the configuration of the data store. The Create, Read, Update, and Delete (CRUD) operations are supported in Oracle NoSQL database. It uses a non-relational data storage system, which does not require a fixed table schema. Oracle NoSQL database is written in Java, and the data can also be accessed using the Java APIs from within other applications. Oracle NoSQL database also supports transparent load balancing, by using an intelligent driver which efficiently distributes the requests among the master and replica nodes. Most of the major data types are supported by Oracle NoSQL database. Unlike other databases, Oracle NoSQL provides us with a very simple administration process.

24.2 Architecture

The data are stored in Oracle NoSQL database in the data store which is called as KVStore. The KVStore is a collection of storage nodes. The storage node hosts one or more number

of replication nodes as determined by its capacity. The store can consist of multiple storage nodes of different capacity. Oracle NoSQL database will intelligently allocate the load as per the capacity of each individual storage node (Oracle, 2011). The replication node consists of at least one partition. There can also be multiple partitions present in the replication node. The data are stored in this replication node. NoSQL DB application code is which invokes the data for processing. Whenever a request arrives, the load balancers and the web servers direct the request to one of the application servers. The application server has the database application code. The application server will be able to communicate with traditional database as well as NoSQL database.

Replication nodes are databases containing key–value pairs. Replication nodes are organized into Shards. The application code interacts with the replication nodes of the Shard. Among the replication nodes in the Shard, one of the replication nodes assumes the role of master. The other replication nodes will become the replicas. Shards are horizontal partitions in database. Each shard contains multiple replication nodes, based on the replication factor. The number of nodes belonging to a shard is called its replication factor.

24.2.1 Read and Write Operations

The read and write operations are performed on the replication nodes. All the database write operations will be performed to the master node from the NoSQL DB Application code. The master node in turn writes the updated data to the replicas. All the database read operations are directed to the replicas. The master node also performs read when required. If the master node fails, any one of the replicas in that shard will be elected automatically as the master node (Oracle, 2016c).

24.2.2 Key–Value Store

Key–value store is a collection of key and value pairs. The key is used to identify a particular data entry uniquely. The value is the data that are pointed by the key. Generally, there are many constraints on keys but very less constraints on the values. There are very limited query mechanisms that are available to perform complex processing on values. The keys are classified into major keys and minor keys according to the hierarchy by which they are represented. Key–value stores are ideal if there are large numbers of binary data blobs like images, which need to be stored once and retrieved many times (Oracle, 2016a).

24.3 KVLite

The KVLite is a single-node, single-shard store which usually runs in a single process. It is a simplified version of Oracle NoSQL database that does not support replication and security. KVLite can be accessed through command-line interface or through drivers written for Java, C, node.js, and Python. KVLite is installed when Oracle NoSQL database is installed (Oracle, 2016c).

24.4 Installation

Download Oracle NoSQL Database Server, Community Edition from Oracle download page in <http://www.oracle.com/technetwork/database/database-technologies/nosqldb>, and unzip it in your local file system.

For example: kv-ce-3.5.2.zip

This is your local version of Oracle NoSQL KVLite Database. Java 7 or later version of Java must be present on your system, and the environment variables must be configured appropriately.

24.5 Getting Started

Open the command prompt, and navigate to kv-ce-3.5.2\kv-3.5.2\lib path. For example,

```
cd c:\ kv-ce-3.5.2\kv-3.5.2\lib
```

The lib folder has the kvstore.jar file, using which the kvlite can be initialized.

Start and initialize KVLite by typing:

```
java -jar kvstore.jar kvlite
```

You can specify `-Xms` and `-Xmx` flags for java to avoid too much heap size. During the first-time execution, the kvlite creates a new store with the following output:

```
Created new kvlite store with args:  
-root ./kvroot -store kvstore -host localhost -port 5000 -admin 5001
```

This will create a new kvlite store, and the nosql data will be placed in `./kvroot`, which is also known as kvroot. The default values for various parameters are set. In the above output, the store name is kvstore, the host name is localhost, the registry port is 5000, and the administration port is 5001.

On consecutive executions, the kvlite opens the existing store with the following output:

```
Opened existing kvlite store with config:  
-root ./kvroot -store kvstore -host localhost -port 5000 -admin 5001
```

Do not close the command prompt as it will close the kvstore. The kvstore must be open throughout the execution.

Open another command prompt and navigate to kv-ce-3.5.2\kv-3.5.2\lib path. Launch the kv shell by typing:

```
java -jar kvstore.jar runadmin -host localhost -port 5000 -store kvstore
```

This will bring up the kv shell where we can type and execute various commands. Type `help`, and then press ENTER to see the list of shell commands. Type `help COMMAND` for help on specific command usage.

Now, let us insert a simple key–value pair data, and then clean up.

To insert a key–value pair of data with key as *testkey* and value as *testvalue*, enter:

```
kv-> put kv -key /testkey -value testvalue
```

Operation is successful, and record is inserted.

This command will store the key and value in the store. Notice the / before the *testkey*. This is required to specify that it is a first-level key.

Optionally, *-if-present* and *-if-absent* can be used for conditional insert in to the database.

To prove that the key–value pair was inserted into the database, run the *get* command. This will output the key–value having key as *testkey* in the store.

```
kv-> get kv -key /testkey
testvalue
```

To see the all the key–value pairs in the database, use the following command. This will display all the keys with their values present in the database, type:

```
kv-> get kv -all
```

To remove the key–value pair from the database, use the delete command. Type:

```
kv-> delete kv -key /testkey
Key deleted: /testkey
```

Exit from the kv shell by typing *exit*. You can stop the *kolite* by typing *ctrl+c* in the command prompt window.

24.6 Tables

We can also create tables in Oracle NoSQL database. Now, let us create a simple table, add some data, and then clean up.

To create a table, the name of the table and the columns in the table must be specified in the table definition. Each column must have a data type, and one of the columns must be the primary key for the table (Oracle, 2016b).

To create a table named *student* with three columns *studentid*, *studentname*, and *coursename* having *studentid* as its primary key, enter:

```
kv-> execute "CREATE TABLE student (
-> studentid INTEGER,
-> studentname STRING,
-> coursename STRING,
-> PRIMARY KEY (studentid))";
Statement completed successfully
```

To prove that the new table was created successfully, run the *show tables* command. This will output all the tables in the database.

```
kv-> show tables;
Tables:
    student
```

To insert three rows of data with three columns into the table, execute the *put* command. Type:

```
kv-> put table -name student -json '{"studentid":101,
"studentname":"jack", "coursename":"physics"}'
Operation successful, row inserted.
kv-> put table -name student -json '{"studentid":102,
"studentname":"bob", "coursename":"maths"}'
Operation successful, row inserted.
kv-> put table -name student -json '{"studentid":103,
"studentname":"peter"}'
Operation successful, row inserted.
```

Notice that the third row of record does not have the *coursename* value. Any such fields that do not have any values will be auto populated with *null*. List the table contents using the following command:

```
kv-> get table -name student
{"studentid":101,"studentname":"jack","coursename":"physics"}
{"studentid":102,"studentname":"bob","coursename":"maths"}
{"studentid":103,"studentname":"peter","coursename":null}
```

A row in the table can also be updated using the *put* command. The *put* command will create a new record if no record exists; however, if any record exists, then the values get updated. Type:

```
kv-> put table -name student -json '{"studentid":101,
"studentname":"jack", "coursename":"Chemistry"}'
Operation successful, row updated.
```

Notice that the *coursename* field of the record with *studentid* as 101 has been changed from *physics* to *chemistry*.

Any row in the table can be deleted using the *delete* command. To drop the record with *studentid* 103 from the database type:

```
kv-> delete table -name student -field studentid -value "103"
1 row deleted.
```

To delete all the rows in the table type:

```
kv-> delete table -name student -delete-all
2 rows deleted.
```

To remove the table completely from the database, do the following:

```
kv-> execute 'DROP TABLE student'
Statement completed successfully
```

To learn how to set up indexes, consistency, durability, etc., see the Oracle NoSQL database, Getting started guide.

24.7 Java Driver

The data in the Oracle NoSQL database KVStore can also be accessed using Java Drivers provided by Oracle. Java Drivers are provided to access both the KV and tables.

24.7.1 Java KV Access

The following example shows the Java version of the shell operations discussed in the previous section.

EXAMPLE 24.1: BASIC KV ACCESS

```
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.Key;
import oracle.kv.Value;
public class KVExample {
    public static void main(String[] args) {
        // connect to kvstore
        KVStore kvStore = KVStoreFactory.getStore(new KVStoreConfig("kvstore",
            new String[] { "localhost:5000" }));
        // create key and value
        Key myKey = Key.createKey("testkey");
        Value myValue = Value.createValue("testvalue".getBytes());
        // put key value pair in the kvstore
        kvStore.put(myKey, myValue);
        System.out.println(myKey + " successfully inserted.");
        // get key value pair from the kvstore
        String returnValue = new String(kvStore.get(myKey).getValue()
            .getValue());
        System.out.println(returnValue);
        // delete a key value pair in the kvstore
        kvStore.delete(myKey);
        System.out.println(myKey + " successfully deleted.");
    }
}
```

In this class, we first get a handler for `oracle.kv.KVStore` using the `oracle.kv.KVStoreFactory` class. Machine name and port number are passed as parameters. The `getStore()` method returns an instance of `oracle.kv.KVStore`. We can create, read, update, and delete using the instance of `oracle.kv.KVStore`. The `createKey()` method of `oracle.kv.Key` class is used to create key and the `createValue()` method of `oracle.kv.Value` class is used to create value. The `KVStore.put()` method is used to insert record into the store. This method will update the record if the key already exists in the store or will create a new record if the key does not exist in the store. The `KVStore.get()` method is used to read value from the store. It takes the key as parameter and returns a byte array. The `KVStore.delete()` method is used to delete a record from the store. It takes the key as input parameter.

24.7.2 Java Table Access

The following example shows the Java version of the shell operations for tables discussed in the previous section.

EXAMPLE 24.2: BASIC TABLE ACCESS

```

import java.util.Map;
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.table.PrimaryKey;
import oracle.kv.table.Row;
import oracle.kv.table.Table;
import oracle.kv.table.TableAPI;
public class KVExample {
    public static void main(String[] args) {
        // connect to kvstore
        KVStore kvStore = KVStoreFactory.getStore(new KVStoreConfig("kvstore",
            new String[] { "localhost:5000" }));
        // Create a table in the store
        String statement1 = "CREATE TABLE student (studentid
            INTEGER, studentname STRING, coursename
            STRING, PRIMARY KEY (studentid))";
        kvStore.executeSync(statement1);
        System.out.println("Table successfully created");
        // Show all the tables in the store
        TableAPI tableAPI = kvStore.getTableAPI();
        Map listTables = tableAPI.getTables();
        System.out.println(listTables.keySet());
        // insert rows into the table.
        Table myTable = tableAPI.getTable("student");
        Row row1 = myTable.createRow();
        row1.put("studentid", 101);
        row1.put("studentname", "jack");
        row1.put("coursename", "physics");
        tableAPI.put(row1, null, null);
        Row row2 = myTable.createRow();
        row2.put("studentid", 102);
        row2.put("studentname", "bob");
        row2.put("coursename", "maths");
        tableAPI.put(row2, null, null);
        Row row3 = myTable.createRow();
        row3.put("studentid", 103);
        row3.put("studentname", "peter");
        tableAPI.put(row3, null, null);
        System.out.println("Successfully inserted 3 records");
        // read the records in the table.
        PrimaryKey myKey1 = myTable.createPrimaryKey();
        myKey1.put("studentid", 101);
        Row myRow = tableAPI.get(myKey1, null);
        String sName = myRow.get("studentname").asString().get();
        String cName = myRow.get("coursename").asString().get();
        System.out.println(sName + " " + cName);
        // delete row from the table
        PrimaryKey myKey2 = myTable.createPrimaryKey();
        myKey2.put("studentid", 103);
        tableAPI.delete(myKey2, null, null);
        System.out.println("Successfully deleted");
        // drop the table in the store
        String statement2 = "DROP TABLE student";
        kvStore.executeSync(statement2);
        System.out.println("Table successfully dropped");
    }
}

```

In this class, the handler for `oracle.kv.KVStore` is fetched using the `getStore()` method. This is required to obtain the `KVStore` connection. The `executeSync()` method is used to execute any commands to the store. `CREATE TABLE` command is used to create the table and is executed using the `executeSync()` method. A table named `student` with three fields, namely, `studentid`, `studentname`, and `coursename` is created. All the tables in the store can be listed using the methods in `oracle.kv.table.TableAPI` class. `getTables()` method returns a `java.util.Map` with the name of all the tables stored as key of the `Map`. The `keySet()` method of `Map` class is used to get the name of all the tables. Insertion can also be performed using the `TableAPI`. The `oracle.kv.table.Row` is used for row creation, and the value is stored using the `put()` method. Then, the `put()` method is used to insert row into the table. Three such rows are inserted into the table, and success message is printed. To read the records, an object of `oracle.kv.table.PrimaryKey` class is created, and the required row's primary key value is stored in it. The `get()` method of `TableAPI` is used to fetch the row. It returns an `oracle.kv.table.Row` object, which contains the record. The `get()` method of `Row` class is used to get individual values of the columns of the table, and they are printed. Delete operation can be performed on the table. An object of `oracle.kv.table.PrimaryKey` class is created, and the required row's primary key value is stored in it. The `delete()` method of `TableAPI` is used to delete the row from the table. Finally, we clean up by dropping the table itself. We use the `DROP TABLE` command and use the `executeSync()` method to execute it.

24.8 Loading Data from CSV

Data stored in CSV format can be uploaded to the Oracle NoSQL database. The following example shows a Java program which will load the CSV file into Oracle NoSQL database and store the contents as table format.

EXAMPLE 24.3: LOADING CSV DATA

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.table.Row;
import oracle.kv.table.Table;
import oracle.kv.table.TableAPI;
public class CSVFileLoader {
    void loadData(String fileName) throws Exception {
        File inputFile = null;
        BufferedReader br = null;
        String aLine = null;
        String[] aLineData = null;
        String[] aHeaderData = null;
        KVStore kvStore = null;
        kvStore = KVStoreFactory.getStore(new KVStoreConfig("kvstore",
            new String[] { "localhost:5000" }));
        inputFile = new File(fileName);
```

```

br = new BufferedReader(new FileReader(inputFile));
aLine = br.readLine();
aHeaderData = aLine.split(",");
String tableName = "studentdata";
TableAPI tableAPI = kvStore.getTableAPI();
Table myTable = tableAPI.getTable(tableName);
if (myTable == null) {
    if (aHeaderData.length > 0) {
        String statement = "CREATE TABLE " + tableName + "("; for
            (int i = 0; i < aHeaderData.length; i++) {
                statement = statement + aHeaderData[i] + " STRING,";
            }
        statement = statement + " PRIMARY KEY (" + aHeaderData[0]
            + ")";
        kvStore.executeSync(statement);
        myTable = tableAPI.getTable(tableName);
    }
}
Row row = null;
String myData = null;
aLine = br.readLine();
while (aLine != null) {
    aLineData = aLine.split(",");
    row = myTable.createRow();
    for (int i = 0; i < aLineData.length; i++) {
        myData = aLineData[i];
        if (myData == null || myData.length() == 0)
    {
        myData = "null";
    }
    row.put(aHeaderData[i], myData);
}
tableAPI.put(row, null, null);
aLine = br.readLine();
}
}
public static void main(String[] args) throws Exception {
    CSVFileLoader loader = new CSVFileLoader();
    loader.loadData("c:\\studentdetails.csv");
    System.out.println("successfully uploaded");
}
}

```

The above example demonstrates the steps required for loading CSV data into the Oracle NoSQL database.

References

- Alam, M., Muley, A., Kadaru, C., and Joshi, A. 2013. *Oracle NoSQL Database: Real-Time Big Data Management for the Enterprise*. Oracle Press, USA.
- Oracle. 2011. Oracle NoSQL Database. Oracle White Paper. Retrieved from <http://www.oracle.com/technetwork/database/database-technologies/nosqlldb/learnmore/nosql-database-498041.pdf>

- Oracle. 2016a. Getting Started with Oracle NoSQL Database Key/Value API. Retrieved from <http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/index.html>
- Oracle. 2016b. Getting Started with Oracle NoSQL Database Tables. Retrieved from <http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/index.html>
- Oracle. 2016c. Oracle NoSQL Database Administrator's Guide. Retrieved from <http://docs.oracle.com/cd/NOSQL/html/AdminGuide/index.html>

25

Hosting and Delivering Cassandra NoSQL Database via Cloud Environments

Skylab Reddy and Pethuru Raj

CONTENTS

25.1	Introduction.....	415
25.2	Big Data Characteristics.....	416
25.3	Big Data Challenges	416
25.3.1	About Big Data Analytics.....	417
25.4	Optimal Infrastructures for Big Data Analytics	418
25.5	Newer and Nimbler Big Data Applications	420
25.6	Describing NoSQL Databases.....	420
25.7	The Major Types of NoSQL Databases	421
25.7.1	Graph Databases.....	421
25.7.2	Key-Value Databases	421
25.7.3	Columnar Databases.....	422
25.7.4	Document Databases	423
25.8	Why NoSQL Databases?	423
25.9	Jumping into the Cassandra NoSQL Database	424
25.10	Setting Up the Cassandra Cluster.....	426
25.11	Using Different Client APIs to Access Cassandra	432
	References.....	433

25.1 Introduction

Apache Cassandra, a leading NoSQL database, supports distributed architecture providing awe-inspiring performance at extreme data velocities. Similarly, the much-expected high availability is being guaranteed. As for availability, Apache Cassandra is a massively scalable NoSQL database. The idea of Cassandra database has actually originated from the extra needs of the well-known Internet companies (Google, Amazon, and Facebook). These companies have to handle a tremendous amount of multistructured data in order to be right and relevant for their consumers. The well-known and widely used social website (Web 2.0) facebook.com has developed and freely contributed it for the open-source community to bring forth additional features, functionalities, and facilities onto the famous Cassandra database. This database is being used today by numerous modern businesses to manage their critical data. This chapter is specially crafted in order to express and expose all that are needed to install Cassandra and use it from a cloud environment.

The cloud storage has significantly brought down the software deployment and data storage costs. For high availability, data could be stored across geographically distributed

and different cloud environments. Then, there is the need to store unstructured data such as social media posts and multimedia files. SQL databases are extremely efficient at storing structured information, but for storing and working with poly-structured data, the traditional SQL databases struggle a lot. The other key trend is that agile development methods demand that the database schema needs to change very often as business and technology [1] domains are undergoing a variety of distinct changes. As we all know, SQL databases require their structure to be specified in advance. That is, if there is any change to be enacted on SQL databases, then it is a tedious and tough assignment indeed.

In response to these changes, new ways of storing data through NoSQL databases have emerged that allow data to be grouped together more naturally and logically. All these demand loosening and lightening the constricting restrictions on the database schema. Let us start with a brief of NoSQL databases and why they are acquiring and attracting a lot of attention these days. Before that, let us discuss something about Big data and the associated challenges with Big data analytics.

25.2 Big Data Characteristics

Big data is the general term used to represent massive amounts of data that are not stored in the relational form in traditional enterprise-scale databases. New-generation database systems are being unearthed in order to store, retrieve, aggregate, filter, mine, and analyze Big data efficiently. The following are the general characteristics of Big data:

- Data storage is defined in the order of petabytes, exabytes, etc. in volume to the current storage limits (gigabytes and terabytes).
- There can be multiple structures (structured, semistructured, and less structured) for Big data.
- Multiple types of data sources (sensors, machines, mobiles, social sites, etc.) and resources for Big data.
- Data are time sensitive (near real time as well as real time). That means Big data consists of data collected with relevance to the time zones so that timely insight can be extracted.

25.3 Big Data Challenges

Since Big data is an emerging domain, there can be some uncertainties, potential roadblocks, and landmines that could probably unsettle the expected progress. Let us consider a few that are more pertinent:

- *Technology*: Technologies and tools are very important for creating the business value of Big data. There are multiple products and platforms from different vendors. However, the technology choice is very important for firms to plan and proceed without any hitch in their pursuit. The tool and technology choices will

vary depending on the types of data to be manipulated (e.g., XML documents, social media, and sensor data), business drivers (e.g., sentiment analysis, customer trends, and product development), and data usage (analytic or product development focused).

- *Data governance*: Any system has to be appropriately governed in order to be strategically beneficial. Due to the sharp increase in data sources, types, channels, formats, and platforms, data governance is an important component in efficiently regulating the data-driven tasks. Other important motivations include data security while in transit and in persistence, data integrity and confidentiality. Furthermore, there are governmental regulations and standards from world bodies, and all these have to fully comply with in order to avoid any kind of ramifications at a later point in time.
- *Skilled resources*: It is predicted by McKinsey Global Institute (MGI) that there will be a huge shortage of human talent for organizations providing Big data-based services and solutions [2]. There will be requirements for data modelers, scientists, and analysts in order to get all the envisaged benefits of Big data. This is a definite concern to be sincerely attended by companies and governments across the world.
- *Accessibility, consumability, and simplicity*: Big data product vendors need to bring forth solutions that extract all the complexities of the Big data framework from users to enable them to extract business value. The operating interfaces need to be intuitive and informative so that the goal of ease of use can be ensured for people using Big data solutions.

Big data's reputation has taken a bit of a battering lately thanks to the allegations that the NSA is silently and secretly collecting and storing people's web and phone records. This has led to a wider debate about the appropriateness of such extensive data-gathering activities. But this negative publicity should not detract people from the reality of Big data. That is, Big data is ultimately to benefit society as a whole. There is more to these massive data sets than simply catching terrorists or spying on law-abiding citizens.

In short, Big data applications, platforms, appliances, and infrastructures need to be designed in a way to facilitate their usage and leverage for everyday purposes. The awareness about the potentials needs to be propagated widely, and professionals need to be trained in order to extract better business value out of Big data. Competing for technologies, enabling methodologies, prescribing patterns, evaluating metrics, key guidelines, and best practices need to be unearthed and made as reusable assets.

25.3.1 About Big Data Analytics

This recent entrant of Big data analytics into the continuously expanding technology landscape has generated a lot of interest among industry professionals as well as academicians. Big data has become an unavoidable trend, and it has to be solidly and succinctly handled in order to derive time-sensitive and actionable insights. There is a dazzling array of tools, techniques, and tips evolving in order to quickly capture data from diverse distributed resources and process, analyze, and mine the data to extract actionable business insights to bring in technology-sponsored business transformation and sustenance. In short, analytics is the thriving phenomenon in every sphere and segment today. Especially with the automated capture, persistence, and processing of the tremendous amount of multistructured data getting generated by men as well as machines, the analytical value, scope, and

power of data are bound to blossom further in the days to unfold. Precisely speaking, data are a strategic asset for organizations to insightfully plan to sharply enhance their capabilities and competencies and to embark on the appropriate activities that decisively and drastically power up their short- as well as long-term offerings, outputs, and outlooks. Business innovations can happen in plenty and be sustained too when there is a seamless and spontaneous connectivity between data-driven and analytics-enabled business insights and business processes.

In the recent past, real-time analytics have gained much prominence and several product vendors have been flooding the market with a number of elastic and state-of-the-art solutions (software as well as hardware) for facilitating on-demand, ad hoc, real-time, and runtime analysis of batch, online transaction, social, machine, operational, and streaming data. There are a number of advancements in this field due to its huge potentials for worldwide companies in considerably reducing operational expenditures while gaining operational insights. Hadoop-based analytical products are capable of processing and analyzing any data type and quantity across hundreds of commodity server clusters [3]. Stream computing drives continuous and cognitive analysis of massive volumes of streaming data with sub-millisecond response times. There are enterprise data warehouses, analytical platforms, in-memory appliances, etc. Data Warehousing delivers deep operational insights with advanced in-database analytics. The EMC Greenplum Data Computing Appliance (DCA) is an integrated analytics platform that accelerates analysis of Big data assets within a single integrated appliance. IBM PureData System for Analytics architecturally integrates database, server, and storage into a single, purpose-built, easy-to-manage system. Then, SAP HANA is an exemplary platform for efficient Big data analytics. Platform vendors are conveniently tied up with infrastructure vendors especially cloud service providers (CSPs) to take analytics to the cloud so that the goal of analytics as a service (AaaS) sees a neat and nice reality sooner than later. There are multiple startups with innovative product offerings to speed up and simplify the complex part of Big data analysis.

25.4 Optimal Infrastructures for Big Data Analytics

There is no doubt that consolidated and compact platforms accomplish a number of essential actions toward simplified Big data analysis and knowledge discovery. However, they need to run in optimal, dynamic, and converged infrastructures to be effective in their operations [4]. In the recent past, IT infrastructures went through a host of transformations such as optimization, rationalization, and simplification. The cloud idea has captured the attention of infrastructure specialists these days as the cloud paradigm is being proclaimed as the most pragmatic approach for achieving the ideals of infrastructure optimization. Hence, with the surging popularity of cloud computing, every kind of IT infrastructure (servers, storages, and network solutions) is being consciously subjected to a series of modernization tasks to empower them to be policy based, software defined, cloud compliant, service oriented, networkable, programmable, etc. That is, Big data analytics is to be performed in centralized/federated, virtualized, automated, shared, and optimized cloud infrastructures (private, public, or hybrid). Application-specific IT environments are being readied for the Big data era. Application-aware networks are the most sought-after communication infrastructures for Big data transmission and processing. [Figure 25.1](#)

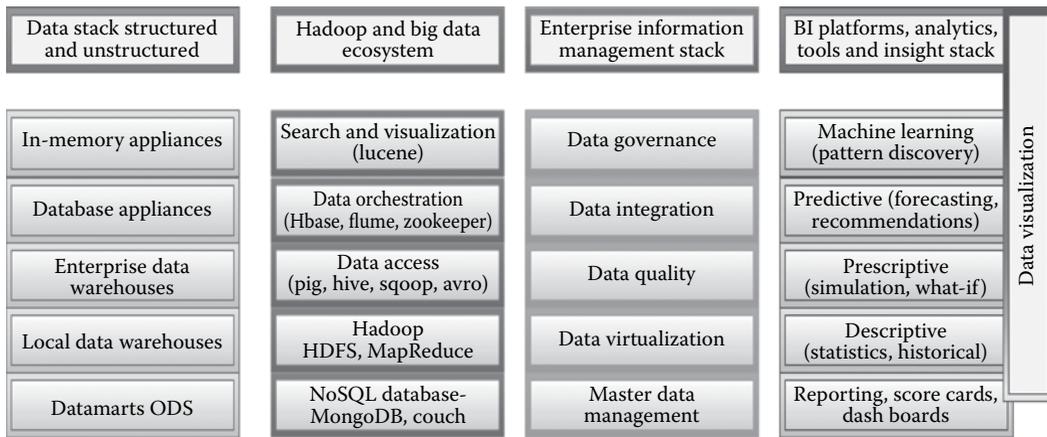


FIGURE 25.1
Big data analytics platforms, appliances, products, and tools.

vividly illustrates all the relevant and resourceful components for simplifying and streamlining Big data analytics.

As with data warehousing, data marts, and online stores, an infrastructure for Big data too has some unique requirements. The ultimate goal here is to easily integrate Big data with enterprise data to conduct deeper and influential analytics on the combined data set. As per the white paper titled “Oracle: Big Data for the Enterprise,” there are three prominent requirements (data acquisition, organization, and analysis) for a typical Big data infrastructure. NoSQL has all these three intrinsically [5].

- *Acquire Big data:* The infrastructure required to support the acquisition of Big data must deliver low and predictable latency in both capturing data and in executing short and simple queries. It should be able to handle very high transaction volumes often in a distributed environment and also support flexible and dynamic data structures. NoSQL databases are the leading infrastructure to acquire and store Big data. NoSQL databases are well suited for dynamic data structures and are highly scalable. The data stored in an NoSQL database are typically of a high variety because the systems are intended to simply capture all kinds of data without categorizing and parsing the data. For example, NoSQL databases are often used to collect and store social media data. While customer-facing applications frequently change, underlying storage structures are kept simple. Instead of designing a schema with relationships between entities, these simple structures often just contain a major key to identify the data point and then a content container holding the relevant data. This extremely simple and nimble structure allows changes to take place without any costly reorganization at the storage layer.
- *Organize Big data:* In classical data warehousing terms, organizing data is called data integration. Because there is such a huge volume of data, there is a tendency and trend gathering momentum to organize data at its original storage location. This saves a lot of time and money as there is no data movement. The brewing need is to have a robust infrastructure that is innately able to organize Big data, process and manipulate data in the original storage location. It has to support very high throughput (often in batch) to deal with large data processing steps and

handles a large variety of data formats (unstructured, less structured, and fully structured).

- *Analyze Big data:* The data analysis can also happen in a distributed environment. That is, data stored in diverse locations can be accessed from a data warehouse to accomplish the intended analysis. The appropriate infrastructure required for analyzing Big data must be able to support deeper analytics such as statistical analysis and data mining on a wider variety of data types stored in diverse systems, to scale to extreme data volumes, to deliver faster response times driven by changes in behavior, and to automate decisions based on analytical models. Most importantly, the infrastructure must be able to integrate analysis on the combination of Big data and traditional enterprise data to produce exemplary insights into fresh opportunities and possibilities. For example, analyzing inventory data from a smart vending machine in combination with the events calendar for the venue in which the vending machine is located will dictate the optimal product mix and replenishment schedule for the vending machine.

25.5 Newer and Nimblers Big Data Applications

The success of any technology is to be squarely decided based on the number of mission-critical applications it could create and sustain. That is, the applicability or employability of the new paradigm to as many application domains as possible is the main deciding factor for its successful journey. As far as the development is concerned, Big data applications could differ from other software applications to a larger extent. Web and mobile enablement of Big data applications are also important. As Big insights are becoming mandatory for multiple industry segments, there is a bigger scope for Big data applications. Therefore, there is a Big market for Big data application development platforms, patterns, metrics, methodology, reusable components, etc.

25.6 Describing NoSQL Databases

A NoSQL database environment is typically non-relational in nature and principally distributed database system that enables the rapid and ad hoc organization and analysis of massive volumes of multistructured data. With Big data and cloud paradigms get accentuated, NoSQL databases are gaining a lot of traction. Sometimes, NoSQL databases are termed as cloud databases. As articulated elsewhere, NoSQL databases are being primed for comfortably storing and working on Big data. The data volumes go up exponentially due to the explosion of data-generating sources. There is a realization among business executives, IT professionals, and academic professors that data are a strategic asset and they cannot be tossed away. Data comprise a lot of actionable insights, and hence these days, corporates and organizations are keen on collecting all kinds of data (internal as well as external) and leveraging them for squeezing out tactic as well as strategic intelligence. Both machine and men-generated data are consciously gathered and stocked in order to be subjected to specialized and deeper investigations at any point of time in order to discover

hidden knowledge and to disseminate the captured to various stakeholders in order to empower them with sufficient and timely information to act upon confidently.

NoSQL databases are good in fulfilling the various nonfunctional requirements (NFRs) and quality of service (QoS) attributes such as scalability, availability, and fault tolerance. These databases have some unique characteristics such as distribution and sharding in order to be right and relevant for the Big data era. The web-scale applications are in need of such kinds of databases in order to fulfill their needs. NoSQL databases have a flexible and schema-less data model. This model intrinsically enables horizontal scalability (scale-out architecture) so that any amount of data can be accommodated. The leverage of hundreds of commodity servers in this new database model comes handy in doing bigger processing yet at the lower costs. The simplicity being supplied by the well-known Hadoop framework works in tandem with these new-generation databases. There are database-specific SQL-like query languages. That is, there are several database–access–interface mechanisms. The contributions of NoSQL databases are really mesmerizing and immense for the Big data days.

25.7 The Major Types of NoSQL Databases

Majorly, there are four types of NoSQL databases. Each has its own capabilities in order to meet up the specific needs of the applications. Different scenarios mandate for different types of NoSQL databases.

25.7.1 Graph Databases

It is becoming increasingly a connected world. Every small or large thing is systematically integrated with one another in order to leverage each other's unique features, facilities, and functionalities. Thus, a graph kind of information representation is going to be the most sought-after and efficient mechanism for futuristic software applications. So, this kind of NoSQL databases strictly follows the graphical representation, which is widely talked in the well-researched graph theory domain. As we all know, graphs typically comprise nodes and edges. Data objects are being indicated by nodes, and the relations among data objects are being represented by edges.

Nodes represent entities such as people, businesses, accounts, or any other item you might want to keep track of. They are roughly the equivalent of the *record*, *relation*, or *row* in a relational database, or the *document* in a document database. Edges are the lines that connect nodes to other nodes. Edges represent the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties, and edges. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in other systems. Properties are pertinent information that relates to nodes. These databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them.

25.7.2 Key–Value Databases

This is a data storage paradigm designed for storing, retrieving, and managing data as dictionaries do. Dictionaries typically contain a collection of *objects* or *records*, which in

turn have many different *fields* within them, each containing data. These records are stored and retrieved using a *key* that uniquely identifies the record and is used to quickly find the data within the database. The picture below vividly illustrates how keys and their values are represented

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

In each key–value pair, the key is represented by an arbitrary string such as a filename, URI, or hash. The value can be any kind of data like an image, user preference file, or document. The value is stored as a blob requiring no upfront data modeling or schema definition. The storage of the value as a blob removes the need to index the data to improve performance. Key–value stores do not have any query language. They provide a way to store, retrieve, and update data using simple *get*, *put*, and *delete* commands. The path to retrieve data is a direct request to the object in memory or on disk. This simplicity enshrined in this model makes it possible to store data fast and easy to use. Furthermore, these databases are highly scalable, portable, and flexible.

Relational databases predefine the data structure in the database as a series of tables containing fields with well-defined data types. Exposing the data types helps any database query to innately apply a number of optimizations. But key–value systems treat the data as a single opaque collection which may have different fields for every record. Because optional values are not represented by placeholders, key–value stores, therefore, consume less memory to store the same database. This memory efficiency comes handy in gaining unprecedented performance advantages.

25.7.3 Columnar Databases

A columnar database is a new-generation database management system storing data in columns rather than in rows. The key driver of a columnar database is to efficiently write and read data to and from hard disks in order to speed up data retrieval. The arrangement here is that all the Column 1 values are physically together followed by all the Column 2 values, etc. The data are stored in the record order, so the 100th entry for Column 1 and the 100th entry for Column 2 belong to the same input record. This allows individual data elements, such as customer name, for instance, to be accessed in columns as a group, rather than individually row by row. Here is an example (<http://searchdatamanagement.techtarget.com/definition/columnar-database>) of a simple database table with four columns and three rows.

ID	Last	First	Bonus
1	Doe	John	8000
2	Smith	Jane	4000
3	Beck	Sam	1000

In a row-oriented database management system, the data would be stored like this: 1,Doe,John,8000;2,Smith,Jane,4000;3,Beck,Sam,1000;

In a column-oriented database management system, the data would be stored like this: 1,2,3;Doe,Smith,Beck;John,Jane,Sam;8000,4000,1000;

This database helps to have data elements and entities highly compressed. The compression permits columnar operations like MIN, MAX, SUM, COUNT, and AVG to be executed quickly. These database implementations enable self-indexing, and hence they consume less disk space.

25.7.4 Document Databases

These simply expand on the basic idea of key–value stores where “documents” contain more complex data objects. Each document is assigned a unique key, and this key is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, which is generally semistructured data. Storing data in this way has some crucial advantages.

Documents are independent units, and hence, the read performance is definitely better and the related data are read contiguously off the disk. Furthermore, this makes it easier to distribute data across multiple servers while preserving its locality. Writing application logic is easy and fast as there is no need for any kind of translations between objects in the application and the SQL queries. Similarly, storing unstructured data is easy since a document contains whatever keys and values the application logic requires. Finally, document databases generally have very powerful query engines and indexing features that make it easy and fast to execute many different optimized queries.

25.8 Why NoSQL Databases?

B2C e-commerce and B2B e-business applications are highly transactional, and the leading enterprise application frameworks and platforms such as Java Enterprise Edition (JEE) directly and distinctly support a number of transaction types (simple, distributed, nested, etc.). For a trivial example, flight reservation application has to be rigidly transactional; otherwise, everything is bound to collapse. As enterprise systems are increasingly distributed, the need for transaction feature is being pronounced as a mandatory one.

In the recent past, social applications have grown fast, and especially youth is totally fascinated by a stream of social computing sites, which has resulted in an astronomical growth of those sites. It is no secret that the popularity, ubiquity, and utility of Facebook, LinkedIn, Twitter, Google+, and other blogging sites are surging incessantly. There is a steady synchronization between enterprise and social applications with the idea of adequately empowering enterprise applications with additional power and value. For example, online sellers understand and utilize customers’ choices, leanings, historical transactions, feedbacks, feelings, etc. in order to do more business. That is, businesses are more interactive, open, and inclined toward customers’ participation to garner and glean their views to reach out to more people across the globe and to pour in Richer Enterprise Applications (REAs). There are specialized protocols and web 2.0 technologies (Atom, RSS, AJAX, mash-up, etc.) to programmatically tag information about people and places and proclivity to dynamically conceive, conceptualize, and concretize more and more people-centric and premium services.

The point to be conveyed here is that the dormant and dumb database technology has to evolve faster in order to accomplish these new-generation IT abilities. With the modern data being more complicated and connected, the NoSQL databases need to have the

implicit and innate strength to handle the multistructured and massive data. A NoSQL database should enable high-performance queries on the data. Users should be able to ask questions such as “Who are all my contacts in Europe?” and “Which of my contacts ordered from this catalog?” A white paper titled as “NoSQL for the Enterprise” by Neo Technology lists out the uniqueness of NoSQL databases for enterprises. I have reproduced the essential things from that paper below.

- *A simplified data representation:* A NoSQL database should be able to easily represent complex and connected data that make up today’s enterprise applications. Unlike traditional databases, a flexible schema that allows for multiple data types also enables developers to easily change applications without disrupting live systems. Databases must be extensible and adaptable. With the massive adoption of clouds, NoSQL databases ought to be more suitable for clouds.
- *End-to-end transactions:* Traditional databases are famous for “all or nothing” transactions, whereas NoSQL databases give a kind of leeway on this crucial property. This is due to the fact that the prime reason for the emergence and evolution of NoSQL databases was to process massive volumes of data in double quick time to come out with actionable inputs. In other words, traditional databases are for enterprise applications, whereas NoSQL databases are for social applications. Specifically, the consistency aspect of ACID transactions is not rigidly insisted in NoSQL databases. Here and there one operation could fail in a social application, and it does not matter much. For instance, there are billions of short messages being tweeted every day, and Twitter will probably survive if a single Tweet is lost. But online banking applications relying on traditional databases have to ensure a very tight consistency in order to be meaningful. That does not mean that NoSQL databases are off the ACID hook. Instead, they are supposed to support ACID transactions including XA-compliant distributed two-phase commit protocol. The connections between data should be stored on a disk in a structure designed for high-performance retrieval of connected data sets, all while enforcing strict transaction management. This design delivers significantly better performance for connecting data than the one offered by relational databases.
- *Enterprise-grade durability:* Every NoSQL database for the enterprise needs to have the enterprise-class quality of durability. That is, any transaction committed to the database will not be lost at any cost under any circumstances. If there is a flight ticket reserved and the system crashes due to an internal or external problem thereafter, when the system comes back, the allotted seat still has to be there. Predominantly, the durability feature is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any software or hardware hitches. Relational databases have employed the replication method for years successfully to guarantee the enterprise-strength durability.

25.9 Jumping into the Cassandra NoSQL Database

Cassandra is the best choice because

- It has the fastest writes among its peers such as HBase and so on
- It is linearly scalable with peer-to-peer design

- No single point of failure
- Read and write requests can be handled without impacting each other's performance
- Handles search queries comprising millions of transactions and lightning-fast speeds
- Fail-safe and highly available with replication factors in place
- Guarantees eventual consistency with the CAP theorem on NoSQL DBs
- Column family design to handle a variety of formats
- No or low licensing cost
- Fewer development ops or operational cost
- It can be extended for integration on a variety of other Big data components

The Apache Cassandra database is definitely the right choice when you need scalability and high availability without impacting the performance. The linear scalability and proven fault tolerance through the leverage of hundreds of commodity hardware or the pay-per-usage cloud infrastructure make it the perfect infrastructure for mission-critical data. Cassandra's support for replicating across multiple datacenters (local or remote) is the best-in-class providing lower latency for your users. Commodity servers are bound to fail frequently, but through redundant servers, the availability of systems is being ensured. Cassandra's data model offers the convenience of column indexes with the performance of log-structured updates. Cassandra provides a strong support for denormalization and materialized views, and powerful built-in caching. In Cassandra database, we create column families where we define the metadata of the columns, but the columns are actually stored as rows. Each row can have different sets of columns, thus making the whole column family relatively unstructured and extendible.

Types of column families: There are two types of column families:

- *Static column family:* As the name suggests, this has a static set of columns and is a very close surrogate of all well-known RDBMS tables, barring a few differences that are a result of its NoSQL heritage.
- *Dynamic column family:* This one gets the true essence of being unstructured and schema less. Here, we do not use predefined columns associated with the column family, but the same can be dynamically generated and supplied by the client application at the time of inserting data into the column family. During the creation or definition of a dynamic column family, we get to define the information about the column names and values by defining the comparators and validators.

Types of columns: There are a variety of columns that Cassandra supports.

- *Standard columns:* These columns contain a name; this is either static or dynamic and set by the writing application. A value (this is actually the attribute that stores the data) and timestamp are shown here: Cassandra makes use of the timestamp associated with the column to find out the last update to the column. When data are queried from Cassandra, it orders by this timestamp and always returns the most recent value.

- *Composite columns*: Cassandra makes use of this storage mechanism to handle clustered rows. This is a unique way of handling all the logical rows together that share the same partition key into a single physical wide row. This enables Cassandra to accomplish the legendary feat of storing two billion columns per row.

Cassandra makes use of the first column defined in the primary key as the partition key; this is also known as the row key.

- *Expiring columns*: These are special types of Cassandra columns that have a time to live (TTL) associated with them; the values stored in these columns are automatically deleted or erased after the TTL has elapsed. These columns are used for use cases where we do not want to retain data older than a stated interval; for instance, if we do not need data older than 24 hours.
- *Counter columns*: These are again specialized function columns that store a number incrementally. They have a special implementation and a specialized usage for situations where we use counters.

25.10 Setting Up the Cassandra Cluster

Cassandra is a very scalable key–value store. It promises eventual consistency, and its distributed ring-based architecture eliminates any single point of failure in the cluster, thus making it highly available. It is designed and developed to support very fast reads and writes over excessively large volumes of data. These fast write and readability make it a very strong contender to be used in an *online transaction processing (OLTP)* application to support large business intelligence systems. Cassandra provides a column family-based data model that is more flexible than typical key–value systems.

We have done the experiment in IBM SoftLayer Cloud. This is the second best and largest public cloud in the world. The readers can find more relevant information on the website www.softlayer.com. SoftLayer gives you the highest performing cloud infrastructure available. One platform takes data centers around the world that are full of the widest range of cloud computing options, and then integrates and automates everything.

- *SoftLayer is a different kind of data center*: We have filled our global data centers with first-class computing, storage, and networking gear. Each location is built, outfitted, and operated the same, so you get the exact same capabilities and availability anywhere in our footprint.
- *Fast, resilient, and seamless around the world*: Our data centers are connected by the industry's most advanced network within a network, which integrates distinct public, private, and internal management networks to deliver lower total networking costs, better access, and higher speed. 2 Tbps between locations. Less than 40 ms of latency into the private network from locations around the world. The SoftLayer API controls everything we offer with more than 3000 documented methods and 180 distinct services. It is the foundation of our own internal management systems, customer portal, and mobile apps. And, we give you complete access, so you can automate your SoftLayer solution to fit your unique needs.

- *Deployment tools:* Scale out and move between servers easily—with bare metal and virtual servers in a unified platform, we can do more with server images than you can imagine. Our Flex Images technology lets you move or scale between bare metal and virtual servers as your resource needs change. Our Image Import and Export tools let you bring your own image to SoftLayer, so you can migrate to SoftLayer with ease. And, Auto Scale lets you automate adding and removing virtual servers with simple, flexible triggers that you create.
- *Web portal and mobile apps:* Manage anything, anytime, anywhere—The beauty of our robust API is that it lets us provide easy-to-use management tools that streamline server administration. Manage any part of your environment through our Web-based customer portal. Or, control your services on the go through powerful mobile apps.

We are going to install an instance of Cassandra in the SoftLayer cloud. All the mandatory information along with the snapshots is given below. If you face any problem, please connect with the authors of this chapter.

cassandrapoc.softlayer.com

Once we logged into server with “root/n0way0ut.”

Check whether java installed or not. In our case, Java is not installed, so we are going to install java with the help of command “yum install java.”

```
[root@cassandrapoc ~]# yum install java
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
--> Package java-1.8.0-openjdk.x86_64 1:1.8.0.65-2.b17.el7_1 will be installed
--> Processing Dependency: java-1.8.0-openjdk-headless = 1:1.8.0.65-2.b17.el7_1 for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: xorg-x11-fonts-Type1 for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjvm.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjli.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjava.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: fontconfig for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
```

```
[root@cassandrapoc ~]# yum install java
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
---> Package java-1.8.0-openjdk.x86_64 1:1.8.0.65-2.b17.el7_1 will be installed
--> Processing Dependency: java-1.8.0-openjdk-headless = 1:1.8.0.65-2.b17.el7_1 for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: xorg-x11-fonts-Type1 for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjvm.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjli.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
--> Processing Dependency: libjava.so(SUNWprivate_1.1)(64bit) for package: 1:java-1.8.0-openjdk-1.8.0.65-2.b17.el7_1.x86_64
```

Now check the java version with help of “java -version”

```

Complete:
[root@cassandrapoc ~]# java -version
openjdk version "1.8.0_65"
OpenJDK Runtime Environment (build 1.8.0_65-b17)
OpenJDK 64-Bit Server VM (build 25.65-b01, mixed mode)
[root@cassandrapoc ~]#

```

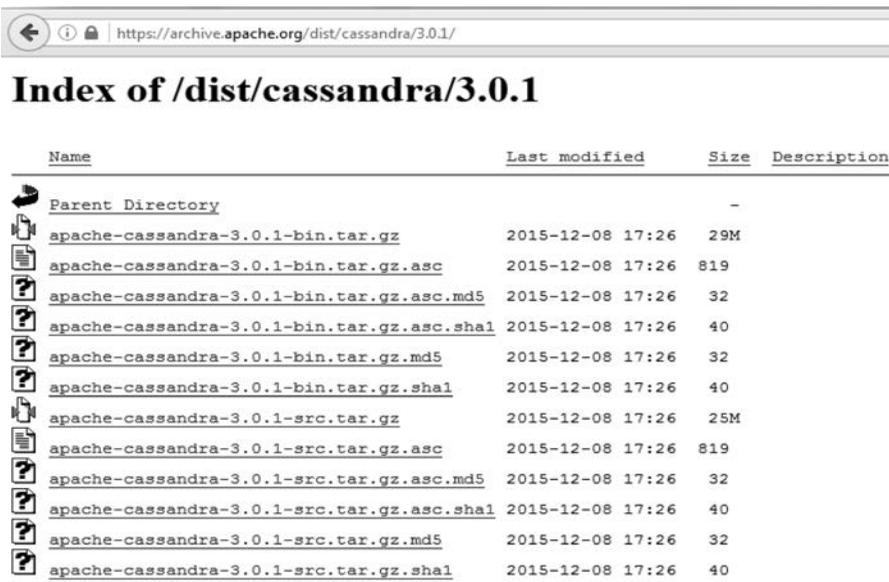
```

[root@cassandrapoc ~]# java -version
openjdk version "1.8.0_65"
OpenJDK Runtime Environment (build 1.8.0_65-b17)
OpenJDK 64-Bit Server VM (build 25.65-b01, mixed mode)
[root@cassandrapoc ~]#

```

Download the “Cassandra” file from the below site and move to server using with “WinSCP” under specified folder structure “cd /usr/local/Cassandra”

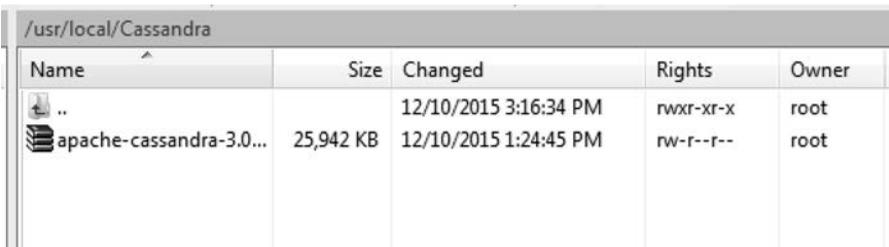
<https://archive.apache.org/dist/cassandra/3.0.1/>



The screenshot shows a web browser window with the address bar containing <https://archive.apache.org/dist/cassandra/3.0.1/>. The main heading is "Index of /dist/cassandra/3.0.1". Below the heading is a table listing files and directories in the directory.

Name	Last modified	Size	Description
Parent Directory		-	
apache-cassandra-3.0.1-bin.tar.gz	2015-12-08 17:26	29M	
apache-cassandra-3.0.1-bin.tar.gz.asc	2015-12-08 17:26	819	
apache-cassandra-3.0.1-bin.tar.gz.asc.md5	2015-12-08 17:26	32	
apache-cassandra-3.0.1-bin.tar.gz.asc.sha1	2015-12-08 17:26	40	
apache-cassandra-3.0.1-bin.tar.gz.md5	2015-12-08 17:26	32	
apache-cassandra-3.0.1-bin.tar.gz.sha1	2015-12-08 17:26	40	
apache-cassandra-3.0.1-src.tar.gz	2015-12-08 17:26	25M	
apache-cassandra-3.0.1-src.tar.gz.asc	2015-12-08 17:26	819	
apache-cassandra-3.0.1-src.tar.gz.asc.md5	2015-12-08 17:26	32	
apache-cassandra-3.0.1-src.tar.gz.asc.sha1	2015-12-08 17:26	40	
apache-cassandra-3.0.1-src.tar.gz.md5	2015-12-08 17:26	32	
apache-cassandra-3.0.1-src.tar.gz.sha1	2015-12-08 17:26	40	

And, move to a server with the help of WinSCP to “/usr/local/Cassandra” directory.



The screenshot shows the WinSCP interface with the local directory path `/usr/local/Cassandra`. The file list is as follows:

Name	Size	Changed	Rights	Owner
..		12/10/2015 3:16:34 PM	rwxr-xr-x	root
apache-cassandra-3.0...	25,942 KB	12/10/2015 1:24:45 PM	rw-r--r--	root

Then give the proper permissions to the specified file with help of “chmod 777 apache-cassandra-3.0.1-src.tar.gz”

```
[root@cassandrapoc Cassandra]# chmod 777 apache-cassandra-3.0.1-src.tar.gz
[root@cassandrapoc Cassandra]#
```

```
[root@cassandrapoc Cassandra]# chmod 777 apache-cassandra-3.0.1-src.tar.gz
[root@cassandrapoc Cassandra]#
```

Then, extract the tar file with help of command “tar zxvf apache-cassandra-3.0.1-src.tar.gz”

```
apache-cassandra-3.0.1-src/bin/sstableloader
apache-cassandra-3.0.1-src/bin/sstablescrib
apache-cassandra-3.0.1-src/bin/sstableupgrade
apache-cassandra-3.0.1-src/bin/sstableutil
apache-cassandra-3.0.1-src/bin/sstableverify
apache-cassandra-3.0.1-src/bin/stop-server
apache-cassandra-3.0.1-src/bin/stop-server.ps1
[root@cassandrapoc Cassandra]#
```

```
apache-cassandra-3.0.1-src/bin/sstableloader
apache-cassandra-3.0.1-src/bin/sstablescrib
apache-cassandra-3.0.1-src/bin/sstableupgrade
apache-cassandra-3.0.1-src/bin/sstableutil
apache-cassandra-3.0.1-src/bin/sstableverify
apache-cassandra-3.0.1-src/bin/stop-server
apache-cassandra-3.0.1-src/bin/stop-server.ps1
[root@cassandrapoc Cassandra]#
```

Set the java path with help of command “set JAVA_HOME=/usr/bin/java”

```
[root@cassandrapoc bin]# set JAVA_HOME=/usr/bin/java
[root@cassandrapoc bin]# java -version
openjdk version "1.8.0_65"
OpenJDK Runtime Environment (build 1.8.0_65-b17)
OpenJDK 64-Bit Server VM (build 25.65-b01, mixed mode)
```

```
[root@cassandrapoc bin]# set JAVA_HOME=/usr/bin/java
[root@cassandrapoc bin]# java -version
openjdk version "1.8.0_65"
OpenJDK Runtime Environment (build 1.8.0_65-b17)
OpenJDK 64-Bit Server VM (build 25.65-b01, mixed mode)
```

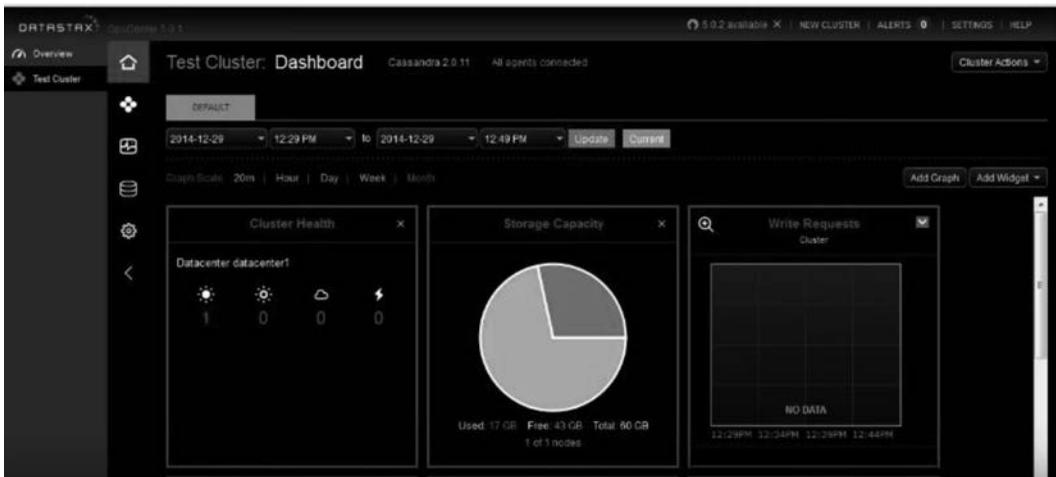
Go to Cassandra directory and start the server with the help of command “cd /usr/dsc-cassandra-2.1.5/bin”

```
[root@cassandrapoc bin]# cd /usr/dsc-cassandra-2.1.5/bin
[root@cassandrapoc bin]# ls
cassandra      cassandra-cli.bat  cassandra.ps1  debug-cql      nodetool.bat    sstablekeys.bat  sstablescrib  sstableupgrade.bat  stop-server.ps1
cassandra.bat  cassandra.in.bat   cqlsh          debug-cql.bat  source-conf.ps1 sstableloader    sstablescrib.bat  stop-server
cassandra-cli  cassandra.in.sh    cqlsh.bat     nodetool       sstablekeys     sstableloader.bat  sstableupgrade  stop-server.bat
```

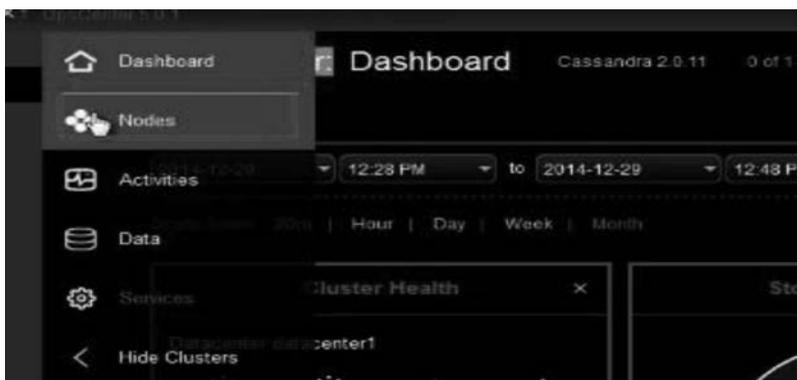
Run the installation command “./cassandra”

```
[root@cassandrapoc bin]# ./cassandra
[root@cassandrapoc bin]# CompilerOracle: inline org/apache/cassandra/db/AbstractNativeCell.compareTo (Ljava/
CompilerOracle: inline org/apache/cassandra/db/composites/AbstractSimpleCellNameType.compareUnsigned (Lorg
CompilerOracle: inline org/apache/cassandra/io/util/Memory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/io/util/SafeMemory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare (Ljava/nio/ByteBuffer;[B)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare ((Ljava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compareUnsigned (Ljava/nio/ByteBuffer;Lja
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/lan
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/lan
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/nio
```

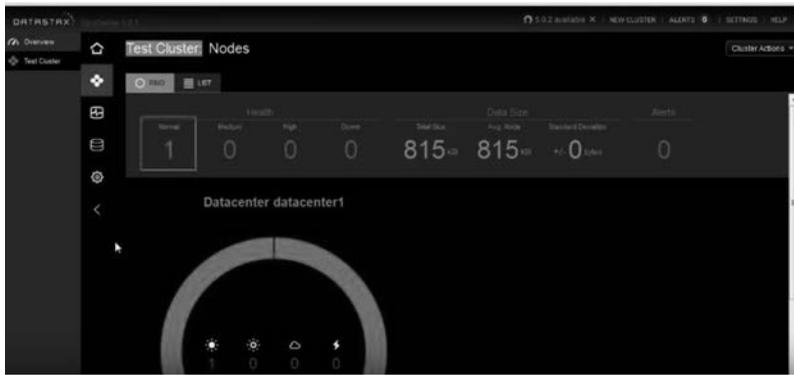
Once installation of Cassandra is completed, we can see the Dashboard like below:



Click on nodes tab.

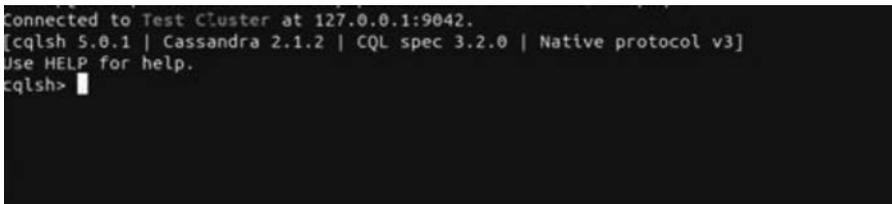


Now, we can see one node because we have not installed any more nodes for now.

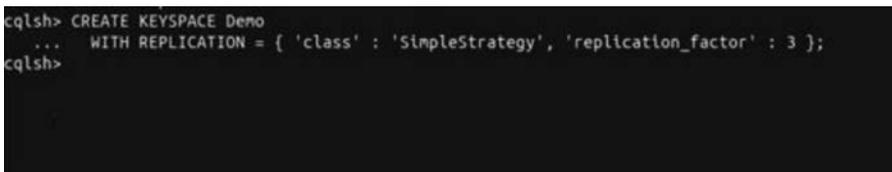


So, as per our installation, we have successfully completed the installation of Cassandra. Now, I am going to run some sample commands, before that I need to enter the Cassandra window using with command “cqlsh”

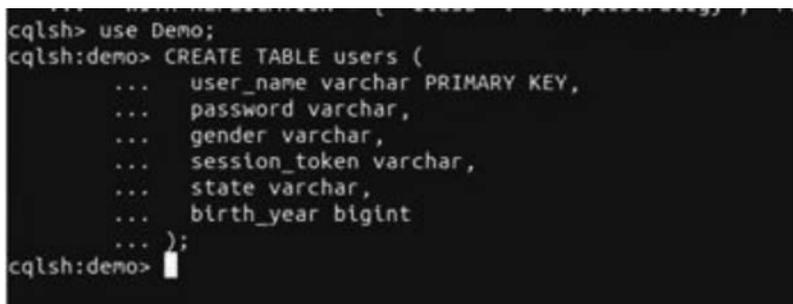
```
[root@cassandrapoc bin]#cd /usr/dsc-cassandra-2.1.5/bin/cqlsh
```



Now I am going to create the database. It is going to be the Cassandra NoSQL database.



As per command, “Keyspace” is created. Now, I am going to create a table under keyspace, and the table name is a user.



Now, we are going to see created table with the help of “select * from system.schema_keyspaces;”

```
cqlsh:deno> select * from system.schema_keyspaces;
```

keyspace_name	durable_writes	strategy_class	strategy_options
system	True	org.apache.cassandra.locator.LocalStrategy	{}
system_traces	True	org.apache.cassandra.locator.SimpleStrategy	{"replication_factor": "2"}
deno	True	org.apache.cassandra.locator.SimpleStrategy	{"replication_factor": "3"}

```
(3 rows)
cqlsh:deno>
```

I am going to describe the table “user” with the “help command”

```
cqlsh:deno> describe table users;
```

```
CREATE TABLE deno.users (
  user_name text PRIMARY KEY,
  birth_year bigint,
  gender text,
  password text,
  session_token text,
  state text
) WITH bloom_filter_fp_chance = 0.01
AND caching = '{"keys": "ALL", "rows_per_partition": "NONE"}'
AND comment = ''
AND compaction = {'min_threshold': '4', 'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32'}
AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
AND dlocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99.0PERCENTILE';
```

25.11 Using Different Client APIs to Access Cassandra

Now that we are acquainted with Cassandra, let us move on to the next step where we will access (insert or update) data into the cluster programmatically. In general, the APIs we are talking about are wrappers written over the core Thrift API, which offers various CRUD operations over the Cassandra cluster using programmer-friendly packages.

The client APIs that are used to access Cassandra are as follows.

Thrift protocol: The most basic of all APIs to access Cassandra is the *Remote Procedure Call (RPC)* protocol, which provides a language-neutral interface and thus exposes flexibility to communicate using Python, Java, and so on. Please note that almost all other APIs we will discuss use *Thrift* under the hood. It is simple to use, and it provides basic functionality out of the box like ring discovery and native access. Complex features such as retry, connection pooling, and so on are not supported out of the box. However, there are a variety of libraries that have extended Thrift and added these much-required features, and we will touch upon a few widely used ones in this chapter.

Hector: This has the privilege of being one of the most stable and extensively used APIs for Java-based client applications to access Cassandra. As mentioned earlier, it uses Thrift under the hood, so it essentially cannot offer any feature or functionality not supported by the Thrift protocol. The reason for its widespread use is that it has a number of essential features ready to use and available out of the box:

- It has implementation for connection pooling
- It has a ring discovery feature with an add-on of automatic failover support
- It has a retry option for downed hosts in the Cassandra ring

Datastax Java driver: This is, again, a recent addition to the stack of client access options to Cassandra and hence goes well with the newer version of Cassandra. Here are its salient features:

- Connection pooling
- Reconnection policies
- Load balancing
- Cursor support

References

1. Bloor, R. 2011. "Enabling the Agile Business with an Information Oriented Architecture," the Bloor Group.
2. McKinsey Global Institute. 2011. Big Data: The Next Frontier for Innovation, Competition, and Productivity. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation
3. Russom, P. 2011. "Hadoop: Revealing Its True Value for Business Intelligence," a White Paper. <http://www.tdwi.org>
4. Oracle. 2011. "Big Data for the Enterprise," a White Paper. <http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf>
5. Neo Technology. 2011. "NoSQL for the Enterprise," a Whitepaper. <http://www.neotechnology.com/tag/nosql/>



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Index

A

- AaaS, *see* [Analytics as a service](#)
- AAIs, *see* [Authentication and Authorization Infrastructures](#)
- ABACs, *see* [Attributes-based access control](#)
- Access control, [81](#), [266](#), [278](#); *see also* [Role-based access control model \(RBAC model\)](#)
 - ABAC model, [246](#), [247](#)
 - CA-RBAC, [246–247](#)
 - column families, [81](#)
 - content-based, [245](#)
 - FGAC, [246](#), [258](#), [259](#)
 - geo-RBAC model, [245](#), [246](#)
 - granular, [253](#)
 - TRBAC model, [244–245](#)
- Access control list (ACL), [182](#), [282](#)
- Access layer, [106–107](#)
- Accumulo*, [169](#)
- ACID, *see* [Atomicity, consistency, isolation, and durability](#)
- ACL, *see* [Access control list](#)
- ACP, *see* [Atomic commitment protocol](#)
- Action nodes, [205](#)
- Admin, [207–208](#)
- Advanced encryption standard (AES), [109](#)
- Aerodynamics, [312](#)
- Aerospike, [311–312](#); *see also* [Hands-on aerospike](#)
 - editions, [313](#)
 - engine, [312](#)
 - installing Aerospike Server, [314–319](#)
 - run Aerospike, [320–322](#)
 - virtual machine, [319–320](#)
- AES, *see* [Advanced encryption standard](#)
- Affero General Public License (APGL), [312](#)
- Agent nodes, [201–202](#)
- Aggregate, [15–16](#), [171–172](#), [202](#)
 - hue, [209](#)
- Agile Development, [48](#)
- Agile Infrastructure (AI), [273](#)
- AI, *see* [Agile Infrastructure](#)
- AlchemyDB, [312](#)
- Algorithm for candidate records identification, [127](#), [128](#)
- Allocation strategy, [5](#)
- Amazon, [288](#)
 - SimpleDB, [76](#)
 - Amazon’s cloud-based Elastic Map-Reduce (AmazonEMR), [194](#), [195](#)
- Analytics as a service (AaaS), [418](#)
- Apache Ambari* project, [162–163](#)
- Apache Cassandra database, [122](#), [415](#); *see also* [Oracle NoSQL database](#)
 - Big Data challenges, [416–418](#)
 - Big Data characteristics, [416](#)
 - Cassandra cluster, [426–432](#)
 - Cassandra NoSQL database, [424–426](#)
 - client APIs to accessing Cassandra, [432–433](#)
 - newer and Nimble Big data applications, [420](#)
 - NoSQL databases, [420–424](#)
 - optimal infrastructures for Big data analytics, [418–420](#)
- Apache Chukwa, [167](#)
- Apache Crux, [164](#)
- Apache Derby, [203](#)
- Apache Flume, [167](#), [174](#), [179](#), [182](#), [183–184](#), [201–202](#)
- Apache Hadoop, [178](#), [179](#)
 - and AAI support, [182–183](#)
 - cluster, [182](#)
 - ecosystem, [178–179](#)
 - Hadoop Map-Reduce, [180–181](#)
 - HDFS, [180](#)
 - projects, [181–182](#)
 - YARN, [181](#)
- Apache HBase, [18](#), [45](#), [64](#), [163](#), [164](#), [168](#), [169](#), [174](#), [181](#), [186](#), [201](#), [203–205](#), [279](#), [282](#), [289](#)
- Apache Hive, [64](#), [163](#), [164](#), [170](#), [172](#), [174](#), [181](#), [182](#), [185–187](#), [195](#), [203–205](#)
- Apache Hue, [163](#)
- Apache Kafka, [163](#), [167](#), [184–185](#)
- Apache License, [188](#), [312](#)
- Apache Mahout, [174](#), [178](#), [186](#), [187](#), [200](#), [209](#)
- Apache MapReduce, [105–106](#), [161](#), [165](#), [171–174](#), [180–181](#), [202–203](#), [208](#), [209](#), [218–219](#), [221](#)
- Apache MLlib, [179](#), [181](#), [188](#)
- Apache Oozie, [163](#), [182](#), [200](#), [204–205](#), [206](#), [209](#)
- Apache Oozie Workflow Scheduler, [163](#)
- Apache Ranger, [207–208](#)
- Apache Software Foundation, [184](#), [203](#), [208](#)
- Apache Spark, [165](#), [208–209](#)
 - machine learning library, [187–180](#)

- Apache Sqoop, 58, 167, 200; *see also* Mongify; Neo4J
 - data mapping to Java or Hive data types, 63–64
 - Free-Form Query Imports and Incremental Imports, 63
 - importing data from MySQL using, 60–61
 - importing specific rows or columns, 62–63
 - installation and configuration, 58–59
 - tool for data migration from MySQL to Hadoop, 58
 - Apache Storm, 165, 167
 - Apache Tez, 208
 - Apache YARN, 181, 208
 - Apache Zookeeper, 163, 205–207
 - APGL, *see* Affero General Public License
 - API, *see* Application Programming Interface
 - Application interface layer, *see* Cloud—Interface layer
 - Application Programming Interface (API), 18, 208
 - client APIs to accessing Cassandra, 432–433
 - using java, 374–375
 - using java script, 372–374
 - using ruby, 374
 - Application requester (AR), 14
 - Application server (AS), 14
 - Application Support Protocol, 14
 - AR, *see* Application requester
 - Arbiter, 89
 - Artificial neural networks, 146
 - AS, *see* Application server
 - Asynchronous transactions, 23–24
 - Atlas, Apache, 184
 - Atomic actions, 12
 - Atomic commitment protocol (ACP), 10–11
 - Atomicity, 17
 - of write operations, 84–85
 - Atomicity, consistency, isolation, and durability (ACID), 7, 16, 23, 151, 272
 - consistency, 311–312
 - in distributed transaction processing, 16
 - and non-relational database, 17–18
 - properties, 16, 23, 209
 - significance of, 16
 - significance of CAP theorem, 18
 - transactions, 424
 - Atomic operations, 356
 - Attack graph, 291, 292
 - algorithms/tools for attack graph analysis, 296–297
 - algorithms/tools for attack graph generation, 293–295
 - analysis, 307–308
 - attack graph-based security metrics, 296
 - attack pattern, 304
 - challenges in, 297
 - data model, 300–303
 - exploitable vulnerabilities, 303–304
 - full attack graph of example network, 307
 - generation, 303
 - generation methods, 292
 - graph databases, 292, 300
 - modeling tool, 292–293
 - simple network configuration, 293
 - Attacks, 275
 - Attributes-based access control (ABACs), 243, 246
 - Attributes, 217
 - encryption, 253
 - relationship methodology, 253
 - Auditing process, 262, 266, 272, 273–274
 - Authentication
 - process, 242, 257–258, 261, 268
 - requirement, 277–278
 - Authentication and Authorization Infrastructures (AAIs), 182–183
 - Authorization, 261–262, 266, 268
 - control, 207
 - Automated tiering, 111–112, 113
 - Availability, 3, 4, 6, 17, 51, 76, 94, 115
- B**
- B2B e-business applications, 423
 - B2C e-commerce applications, 423
 - Baseline approach, 129
 - Baseline hashing-based approach, 128–129
 - Basic Storage, 108
 - Basic Timestamp Ordering (BTO), 8–9
 - “before_save” method, 56
 - Berkeley Software Distribution (BSD), 355
 - Berkley DB, 286
 - Big Data, 48, 75, 116, 143–144, 152, 159, 178, 199–200, 272, 275, 287
 - analytics, 417–418
 - in areas of biological research, 154
 - bioinformaticians for potential benefits, 156–157
 - challenges, 416
 - characteristics, 49, 416
 - clinical research, 155
 - environments, 273
 - genomic research, 155–156
 - information and knowledge generation and management, 179

- infrastructures based on IVIS4BigData
 - reference model, 190–192
- newer and Nimbler Big data applications, 420
- optimal infrastructures for analytics, 418–420
- phenomenon, 120
- process model based on CRISP4BigData
 - reference model, 192–193
- proteomic research, 156
- Big data management tools for Hadoop
 - Apache flume, 201–202
 - Apache HBase, 204, 205
 - Apache Hive, 203–204
 - Apache Mahout, 209
 - Apache Oozie, 204–205, 206
 - Apache Pig, 202–203
 - Apache Ranger, 207–208
 - Apache Spark, 208–209
 - Apache Sqoop, 200
 - Apache Tez, 208
 - Apache Zookeeper, 205–207
 - Cassandra, 209–212
 - export tool, 201
 - Hadoop ecosystem, 200
 - Hue, 209
 - import tool, 200–201
- Biological data(base), 143, 145
 - domain and mining, 146–147
 - importance, 147–148
 - types, 147
- Biological DBMS, 148
 - challenges, 152–153
 - hierarchical DBMS, 149
 - languages in DBMS, 150–152, 153
 - models, 149
 - network, 149, 150
 - NoSQL in biology, 153–154
 - object oriented, 149–150
 - relational, 150
- Biological research, Big Data and
 - NoSQL in different areas of, 154–157
- Biometric authentication, 248
- BOW dataset, 222, 230, 231
- BSD, *see* [Berkeley Software Distribution](#)
- BTO, *see* [Basic Timestamp Ordering](#)
- Burrows-Wheeler Transform (BWT), 171
- C**
- CA-RBAC model, *see* [Context-aware role-based access model](#)
- CAP theorem, *see* [Consistency, Availability, Partition tolerance theorem](#)
- Cassandra, 209, 279, 280–281, 323–329
 - architecture, 209–210
 - CURD operations, 212
 - data model, 210
 - links of NoSQL Cassandra, 323–329
 - NodeTool, 330–331
 - Shell commands, 211
 - tools and utilities provided by Cassandra, 329
- Cassandra databases, 210, 259
 - security weaknesses of, 263–265
- Cassandra Query Language (CQL), 210, 264, 280, 330
 - data definition commands, 211–212
- Cassandra query language shell (cqlsh), 210, 330
 - data definition commands, 211–212
- Centralized security administration, 207
- Centralized strategy, 5
- Centroid selection module, 225
 - for parallel K-Means++, 226, 227
 - for serial K-Means++, 226, 227
- Channel, 183–184
- Chukwa (Apache), 167
- Chunks, 86, 161
- CISCO, 49–50
- Citrusleaf (Aerospike), 312
- CLI, *see* [Command Line Interface](#)
- Client-node communication, 259
- Client APIs to accessing Cassandra, 432–433
- Cloud, 99–100, 120
 - architectures, 49
 - cloud computing-based approaches, 125
 - databases, 420–421
 - data management system, 114
 - interface layer, 107
 - services, 193
- Cloudant Local, 333
- Cloudant Query, 336
- Cloud-based Hadoop systems and services, 193–194
- Cloud Computing, *see* [Cloud](#)
- Cloudera Hadoop, 199–200
- Cloudera Impala, 170
- Cloud execution models, 100
 - data storage technology and algorithms in cloud, 103–109
 - distributed data storage algorithms and efficient data storage strategies, 109–114
 - task scheduling in cloud, 101–103

- Cloud paradigm, 100
 - cloud execution models, 100–114
 - NoSQL and, 100
 - performance management in NoSQL, 114–116
- Cloud service providers (CSPs), 418
- Cloud storage architecture, 106
 - access layer, 106–107
 - cloud interface layer, 107
 - data management layer, 107–108
 - storage layer, 108–109
- Cluster assignment module, 225
- Clustering algorithm, 216, 221
 - centroid selection for parallel K-Means++, 226, 227
 - centroid selection for serial K-Means++, 226, 227
 - cluster assignment module on RHadoop, 228, 229
 - dataset, 230
 - distributed programming paradigm, 218–221
 - experimental setup, 230
 - fuzzy clustering algorithms, 217–218
 - hard partitional clustering algorithms, 217
 - hierarchical clustering algorithms, 217
 - iterative clustering module design on RHadoop, 226–228
 - K-means algorithm, 228–229
 - K-means clustering algorithm
 - improvisations, 221–225
 - performance evaluation, 230–235
 - proposed methodology, 225
 - results, 228
 - technology background, 216–217
- Clusters, 233–234
 - management and monitoring tool categorization, 163
 - validation, 216
- CM, *see* Constant memory
- Collaborative mining, 248
- Collector node, 201–202
- Collision management, 132
- Columnar databases, 422–423
- Column-oriented databases, 169, 254
 - management system, 423
- Column(s), 81
 - counter, 426
 - families, 81
 - family stores database, 18, 19
- Combine phase, 133
- Command Line Interface (CLI), 187, 349
- Commercial off-the-shelf components (COTS components), 120
- Commit phase, 24–25
- Commit request phase, 24
- Community Cloud, 194
- Compiler translates queries, 187
- Complete replication strategy, 5
- Component-based data visualization, 188
- Composite
 - columns, 426
 - databases, 147
- Concurrency control in distributed database, 6–9
- Consistency, 17
 - CouchDB, 94–96
 - distributed, 94
 - MongoDB, 88–89
 - SimpleDB, 79–80, 88, 94
- Consistency, Availability, Partition tolerance theorem (CAP theorem), 14, 17, 18
- Constant memory (CM), 123
- Constrained RBAC, 239
- Constrains, 391
- Content-based access control, 245
- Context-aware role-based access model (CA-RBAC model), 246–247
- Control
 - characters, 79
 - concurrency, 6–9
 - nodes, 205
 - object, 32
- Coordinator, 10
- CORBA protocol, 272
- Core modules, 160–162
- COTS components, *see* Commercial off-the-shelf components
- CouchDB, 90, 279, 281–282; *see also* MongoDB
 - architecture, 90
 - consistency, 94–96
 - data model, 90
 - group databases, 91
 - local databases, 91
 - peers, 90
 - remote databases, 91
 - replication and failure handling, 96
 - sharding, 92–93
 - shifts, 91
 - user/feed, 91
 - user/private, 91
 - user/public, 91
- Counter columns, 426
- CPU, 135, 313
- CQL, *see* Cassandra Query Language

- cqlsh, *see* [Cassandra query language shell](#)
 - CREATE command, [390](#)
 - Create, Read, Update, and Delete operations (CRUD operations), [379](#), [405](#)
 - Credit card processing systems, [262–263](#)
 - CRISP4BigData reference model, *see* [Cross Industry Standard Process for Big Data reference model](#)
 - CRISP-DM, *see* [Cross Industry Standard Process for Data Mining](#)
 - Cron job scheduler, [95](#)
 - Cross Industry Standard Process for Big Data reference model (CRISP4BigData reference model), [192–193](#)
 - Cross Industry Standard Process for Data Mining (CRISP-DM), [192](#)
 - Cross-organizational processing methodologies, [190](#)
 - Big Data analysis infrastructures based on IVIS4BigData reference model, [190–192](#)
 - Big Data analysis process model based on CRISP4BigData reference model, [192–193](#)
 - cloud-based Hadoop systems and services, [193–194](#)
 - CRUD operations, *see* [Create, Read, Update, and Delete operations](#)
 - Crux (Apache), [164](#)
 - CSPs, *see* [Cloud service providers](#)
 - CSV, [58](#), [61](#), [66](#), [69](#)
 - LOAD CSV clause, [65](#)
 - loading books from, [389–390](#)
 - loading data from, [412–413](#)
 - CURD operations, [212](#)
 - Cypher, [64](#), [299](#)
 - clauses, [385](#)
 - Cypher Query Language, [69](#), [299–300](#), [304](#), [382–384](#)
- D**
- DAG scheduling, *see* [Directed Acyclic Graph scheduling](#)
 - Data; *see also* [Big Data](#)
 - abstraction, [239](#)
 - access and retrieval tool, [163–164](#)
 - access efficiency in cloud, [110](#)
 - centers, *see* [Large-scale computing infrastructures](#)
 - classification, [113](#)
 - complexity, [154](#)
 - compression techniques, [114](#)
 - confidentiality, [272](#)
 - data-driven information services, [195](#)
 - data-driven knowledge extraction services, [195](#)
 - deduplication techniques, [113–114](#)
 - domain, [146](#)
 - element encryption, [263](#)
 - encoding, [127](#)
 - event, [201](#)
 - filtering/querying and reporting, [187](#)
 - hash-based approach for data encoding, [136](#)
 - ingestion tool, [166–167](#)
 - insight and effectuation, [194](#)
 - integration and collection based on
 - Hadoop, [183–184](#)
 - integrity, [272](#)
 - KB-DSS, [194](#)
 - management and curation based on
 - Hadoop, [184–185](#)
 - management layer, [107–108](#)
 - matching, [248](#)
 - mining, [146](#)
 - in motion protection, [258–259](#)
 - partitioning, [85](#)
 - reduction techniques, [112–113](#)
 - at rest protection, [258–259](#)
 - security, [109](#)
 - service layer, *see* [Cloud—Interface layer](#)
 - use and performance, [85](#)
 - variety, [154](#)
 - visualization, [146](#)
 - volume, [154](#)
 - Data analytics with Hadoop
 - analysis and indexing, [186](#)
 - Apache Mahout, [187](#)
 - Apache spark’s machine learning library, [187–180](#)
 - data filtering/querying and reporting, [187](#)
 - machine learning tools and techniques, [187](#)
 - Database; *see also* [Distributed databases](#); [Graph database](#)
 - encryption, [275](#)
 - firewalls, [275](#)
 - group, [91](#)
 - security, [272](#)
 - sharding, [82](#)
 - technologies, [167–170](#)
 - vendors, [277](#)
 - Database as a server (DBaaS), [333](#)
 - Database management system (DBMS), [1](#), [148](#), [194](#), [277](#)
 - parallel, [3–4](#)
 - Database manager (DM), [10–11](#)

- Database server (DS), [2](#), [14](#)
 - [H3](#), [293](#)
- Data Cleansing, [192–193](#)
- Data Computing Appliance (DCA), [418](#)
- Data control language (DCL), [151](#)
- Data Definition Language (DDL), [151](#)
- Data-driven applications, [311–312](#)
- Data-driven information services, [195](#)
- Data-driven knowledge extraction services, [195](#)
- Data Enrichment process, [193](#)
- Data manager (DM), [10](#)
- Data Manipulation Language (DML), [151](#)
- Data migration techniques
 - [CISCO](#), [49–50](#)
 - [Dice.com](#), [50–51](#)
 - [eBay](#), [51](#)
 - Google cloud dataflow for pipelining data
 - across storage systems, [69–70](#)
 - independent tools or solutions for data migration, [70](#)
 - from MySQL to Hadoop, [58–64](#)
 - RDBMS, [48–49](#)
 - SQL to MongoDB, [52–58](#)
 - SQL to Neo4j, [64–69](#)
 - Walmart, [50](#)
- Data model, [76](#), [77](#), [209](#)
 - attributes, [78](#)
 - constraints on SimpleDB items, [78–79](#)
 - generation of unique IDs by developer, [78](#)
 - Google's BigTable, [80–82](#)
 - items, [78](#)
 - MongoDB, [83–85](#)
 - SimpleDB domain constraints, [77–78](#)
 - values, [79](#)
- Data Node software, [219–220](#)
- Data processing, [164](#)
 - engines, [164–165](#)
 - systems, [76](#)
- Data-pruning stage, [121](#)
- Dataset, [230](#)
- DataStax Cassandra Community Server, [329](#)
- DataStax Distribution of Apache Cassandra (DDC), [325](#)
- Datastax Java driver, [433](#)
- DataStaxOpsCenter Agent, [329](#)
- DataStaxOpsCenter Community, [329](#)
- Data storage, [166](#), [416](#)
 - cloud storage architecture, [106–109](#)
 - database technologies, [167–170](#)
 - data ingestion, [166–167](#)
 - HDFS, [166](#)
 - programming paradigm for cloud applications, [103–106](#)
 - SQL-on-hadoop, [170](#)
 - technology and algorithms in cloud, [103](#)
- DBaaS, *see* [Database as a server](#)
- db.createRole() method, [242–243](#)
- db.createUser() command, [241–242](#)
- DBMS, *see* [Database management system](#)
- db.revokeRolesFromUser() method, [243](#)
- DCA, *see* [Data Computing Appliance](#)
- DCL, *see* [Data control language](#)
- DDBMS, *see* [Distributed database management system](#)
- DDC, *see* [DataStax Distribution of Apache Cassandra](#)
- DDL, *see* [Data Definition Language](#)
- DDM architecture, *see* [Distributed data management architecture](#)
- Decision trees methods, [146–147](#)
- Defense approaches, [292](#)
- Delete command, [389](#)
- DELETE query, [385](#)
- Demission support systems (DSS), [194](#)
- Denial of Service, [284](#)
- Denormalization, [385–386](#)
- Denormalized data models, [84](#)
- Deployment tools, [427](#)
- DF, *see* [Document frequency](#)
- Diabetes dataset, [230](#)
- Dice.com, [50–51](#)
- Directed acyclic graphs, [165](#)
- Directed Acyclic Graph scheduling (DAG scheduling), [100](#), [102–103](#), [163](#), [182](#), [205](#); *see also* [List scheduling](#)
- Directed graph, [378](#)
- DISTINCT filters, [391](#), [392](#)
- Distributed 2PL, [7](#)
- Distributed computing, [100](#)
- Distributed concurrency control, [6](#)
- Distributed consistency, [94](#)
- Distributed coordination service, [181](#)
- Distributed database management system (DDBMS), [2](#), [3–4](#), [251–252](#)
 - promises of, [9–10](#)
- Distributed databases, [1](#), [251](#); *see also* [Database; Graph database](#)
 - comparison of relational database security and NoSQL security, [255–256](#)
 - concurrency control in, [6–9](#)
 - distributed processing and, [2–3](#)
 - NoSQL security, [252–255](#), [260–263](#)
 - parallel DBMS and DDBMS, [3–4](#)
 - promises of DDBMS, [9–10](#)

- securing NoSQL applications, 265–266
- security in, 251–252
- security threats with NoSQL databases, 256–260
- security weaknesses of Cassandra and MongoDB, 263–265
- security weakness in typical NoSQL database, 266–269
- techniques, 4–6
- Distributed data management architecture (DDM architecture), 14
- Distributed data storage algorithms, 109
 - data access efficiency in cloud, 110
 - data security, 109
 - reliability, 109–110
- Distributed environment, 256–257
- Distributed framework, 218
- Distributed Optimistic (OPT), 9
- Distributed processing and distributed database, 2–3
- Distributed programming paradigm, 218
 - Hadoop architecture, 219–220
 - MapReduce programming model, 218–219
 - RHadoop platform, 221
- Distributed relational database architecture (DRDA), 14
- Distributed system, 94
- Distributed transaction, 23
 - support in EJB and MTS, 31–38
- Distributed transaction processing (DTP), 10, 24; *see also* X/Open Distributed Transaction Processing
 - atomic actions and flat transactions, 12
 - distributed database, 1–10
 - EJB, 25–30
 - indexed services of XA interface in, 38–45
 - models, 11
 - MTS model, 30–31
 - nested transactions, 12–13
 - in non-relational database, 15–16
 - NoSQL in, 18–21
 - operations, 10
 - in relational database, 14–15
 - return of ACID property in, 16–18
 - security issues in, 21–22
 - structure of flat transactions, 12
 - technologies in, 25
 - transactions in distributed database system, 11
- Distributed wound-wait locking algorithm, *see* Wound-Wait (WW)
- DM, *see* Data manager; Database manager
- DML, *see* Data Manipulation Language
- Document
 - databases, 168–169, 423
 - document-based NoSQL databases, 254
 - document-oriented databases, 121
 - growth, 85
 - store database, 18
 - structure, 83
- Document frequency (DF), 122
 - update, 127
- Domain, 77, 146
 - constraints, 77–78
 - meta-model, 390
 - modeling, 387
- DRDA, *see* Distributed relational database architecture
- DS, *see* Database server
- DSS, *see* Demission support systems
- DTP, *see* Distributed transaction processing
- Durability, 17
- Dynamic column family, 425
- Dynamic data model, 323
- Dynamic scheduling, 101
- E
- eBay, 51
- Ecosystem cross-domain processing
 - methodologies, 190
 - Big Data analysis infrastructures based on IVIS4BigData reference model, 190–192
 - Big Data analysis process model based on CRISP4BigData reference model, 192–193
 - cloud-based Hadoop systems and services, 193–194
- Edges, 294, 298, 378
- Efficient data storage strategies in cloud, 109, 110; *see also* Data storage
 - automated tiering, 111–112, 113
 - data compression techniques, 114
 - data deduplication techniques, 113–114
 - data reduction techniques, 112–113
 - thin provisioning, 111
- EHR, *see* Electronic health records
- EJB, *see* Enterprise Java Bean
- EJBObject, 26
- Elasticsearch, 273
- Electronic health records (EHR), 155
- Elgamal homomorphic encryption, 286
- Embedded data, 84
- EMC Greenplum Data Computing Appliance, 418

- Encryption, [109](#), [262](#), [266](#), [278](#)
 - End-to-end transactions, [424](#)
 - Energy analytics, [195](#)
 - Enterprise-grade durability, [424](#)
 - Enterprise Cassandra database, [273](#)
 - Enterprise Java Bean (EJB), [25](#)
 - architecture, [26](#)
 - comparison with MTS and, [38](#)
 - distributed transaction support in, [31](#)
 - implementation in NoSQL, [27](#)
 - integrating web services, [28–29](#)
 - NoSQL integration using Hadoop, [29–30](#)
 - transactional model in EJB, [31–34](#)
 - Entities, [20](#)
 - Entity–relationship diagram (ER diagram), [385](#)
 - ER model for social contacts application, [386](#)
 - Ephemeral znode, [207](#)
 - ER diagram, *see* [Entity–relationship diagram](#)
 - ETL, *see* [Extract, transform, and load](#)
 - Event data, [201](#)
 - Execution Engine, [203](#)
 - Execution models in cloud, [101](#)
 - Expiring columns, [426](#)
 - Exploit dependency attack graph, [295](#), [301](#), [302](#)
 - Exploit dependency graph, [294–295](#)
 - Exploit nodes, [295](#)
 - External constraints, [21–22](#)
 - Extract phase, [132–133](#)
 - Extract, transform, and load (ETL), [164](#), [181](#), [186](#)
 - functions, [203](#)
 - operations, [48](#)
 - tools, [202](#)
- F**
- Facebook Presto*, [170](#)
 - Factory Wrapper Object, [26](#)
 - Failure handling
 - CouchDB, [96](#)
 - Google’s BigTable, [82–83](#)
 - MongoDB, [89](#)
 - Falcon (Apache), [182](#), [184](#)
 - Feature extraction process, [216](#)
 - Feature selection process, [216](#)
 - FGAC, *see* [Fine-grained access control](#)
 - Field-level security, [258](#)
 - FIFO, *see* [First-In-First-Out](#)
 - Fine-grained access control (FGAC), [240–241](#), [246](#), [258](#)
 - Firewall-2, [293](#)
 - Firewalls, [266](#)
 - First-come-first-serve algorithm, [101–102](#)
 - First-In-First-Out (FIFO), [102](#)
 - Fixed schema, [90](#)
 - Flat
 - provisioning, [111](#)
 - RBAC, [238](#)
 - transactions, [12](#)
 - Flex Images technology, [427](#)
 - Flume (Apache), [167](#), [174](#), [179](#), [182](#), [183–184](#), [201–202](#)
 - F-measure, [135](#), [138](#)
 - FOREACH query, [385](#)
 - “Foreign Key”, [52](#)
 - Forming signature sets, [127](#)
 - Fragmentation, [5](#)
 - Fragmented strategy, [5](#)
 - Fraud analytics, [394–397](#)
 - Free-Form Query Imports, [63](#)
 - Fuzzy clustering algorithms, [217–218](#)
- G**
- General-purpose graphics processing units (GPGPUs), [120](#), [123](#)
 - parallel algorithms for signature selection on, [132–134](#)
 - Generation of Unique IDs (GUIDs), [78](#)
 - Genetic algorithms, [147](#)
 - Genomic research, [155–156](#)
 - Geo-RBAC model, *see* [Spatial and location-based access model](#)
 - Geospatial data, [393–394](#)
 - Geospatial function, [333](#)
 - Get operation, [83](#)
 - getStore() method, [410](#)
 - Google
 - cloud dataflow for pipelining data across storage systems, [69–70](#)
 - file system, [180](#)
 - Pregel, [379](#)
 - Google’s BigTable, [80](#), [122](#), [209](#)
 - data model, [80–82](#)
 - replication and failure handling, [82–83](#)
 - Sharding, [82](#)
 - GPGPUs, *see* [General-purpose graphics processing units](#)
 - GPUs, [123](#), [127](#), [128](#), [135](#)
 - Granular access process, [272](#)
 - Graph database, [20](#), [254](#), [292](#), [297](#), [421](#); *see also* [Database](#); [Distributed databases](#); [MongoDB](#)
 - attack graph analysis, [307–308](#)
 - attack graph and, [300](#)
 - attack graph data model, [300–303](#)

- attack graph generation, 303–307
- cypher query language, 299–300
- graph query languages, 298–299
- management system, 298
- Neo4j, 299–300
- NoSQL databases, 297–298
- state of, 298
- Graphs, 378; *see also* Neo4j
 - algorithms, 208–209
 - brainstorming, 378
 - databases, 378, 379
 - data models vs., 379–382
 - directed graph, 378
 - graph-based analytics, 396–397
 - graph-based databases, 169
 - GraphX, 209
 - modeling for social contacts application, 386–388
 - query languages, 292, 298–299
 - search, 394
 - social network, 380
 - theory, 378
- Gremlin query language, 299
- Group databases, 91
- GUIDs, *see* Generation of Unique IDs
- H**
- Hadoop, 199–200, 203
 - architecture, 219–220
 - clusters, 205–206
 - components, 209
 - file system, 178–179, 275
 - framework, 265
 - hadoop-based analytical products, 418
 - MapReduce, 161, 180, 219
 - NoSQL integration using, 29–30
 - platform, 207
 - software library, 179–180
 - YARN, 161, 180
- Hadoop distributed file system (HDFS), 160, 161, 166, 180, 201, 203, 209, 218, 219–220
- Hadoop ecosystems, 160, 200
 - Apache Hadoop, 179–181
 - Apache Hadoop and AAI support, 182–183
 - Apache Hadoop projects, 181–182
 - applications of Hadoop tools, 170–174
 - archival and preservation, 185–186
 - and core modules, 160–162
 - data access and retrieval, 163–164
 - data analytics, 186–188
 - data insight and effectuation, 194–195
 - data integration and collection, 183–184
 - data management and curation, 184–185
 - data processing, 164–165
 - data storage, 166–170
 - ecosystem cross-domain and cross-organizational processing methodologies, 190–194
 - external tool and component-based data visualization, 188
 - future projects and directions, 174
 - Hadoop architecture, modules, and projects, 179
 - information and knowledge generation and management in age of big data, 179
 - interaction with visual data views, 189
 - interaction with visualization pipeline and transformation mappings, 189–190
 - IoTs, 159
 - management and monitoring tools, 162–163
 - tools, 162
 - visualizations, 188
- Hadoop In Hadoop Out (HIHO), 167
- Hadoop Process Definition Language (HPDL), 182
- Hadoop tools, 160
 - applications, 170
 - biology and medicine, 171–172
 - material science research, 172–174
- Hadoop user environment (HUE), 163
- Hands-on aerospike; *see also* Aerospike
 - aerospike editions, 313
 - benefits, 312–313
 - create Aerospike virtual machine, 319–320
 - download and setup, 313–314
 - features, 312
 - hardware requirements, 313
 - install Aerospike Server, 314–319
 - installation guide, 313
 - run Aerospike, 320–322
- Hands-on Cassandra for Windows
 - Apache Cassandra is repairing, 328
 - CQL Shell, 330
 - DDC-325,
 - installation, 326, 327
 - install Cassandra, 324
 - links of NoSQL Cassandra, 323
 - NodeTool, 330–331
 - tools and utilities provided by Cassandra, 329
- Hands-on Cloudbant,
 - Cloudbant database, 334
 - Cloudbant Query, 336
 - Creation of document, 335
 - description of return codes, 340

- Hands-on Cloudbant (*Continued*)
 - geospatial indexes, [337](#)
 - IBM Cloudbant NoSQL Database
 - Capabilities, [333](#)
 - structure of delete command, [338–339](#)
 - Hands-on InfluxDB
 - CLI, [349](#)
 - databases using query, [350](#)
 - enable file system caching, [344](#)
 - End-User License Agreement, [346](#)
 - GNU General Public License, [342](#)
 - installation, [347](#)
 - line ending conversions, [345](#)
 - parameters in influx database, [353](#)
 - querying data using HTTP API, [341](#)
 - retention policy using query, [351](#)
 - Start Menu folder, [343](#)
 - type of literals used in influx database, [352](#)
 - Urlgen, [348](#)
 - Hands-on Redis
 - advantages of Redis, [356](#)
 - Browse button to place program's shortcut, [358](#)
 - increment and decrement values, [363](#)
 - installation process, [359](#)
 - License Agreement screen, [357](#)
 - Redis server, [360–362](#)
 - Hard disk drives (HDDs), [166](#)
 - Hard partitional clustering algorithms, [217](#)
 - Hash
 - baseline hashing-based approach, [128–129](#)
 - collision, [123](#)
 - hash-based approach for data encoding, [136](#)
 - hash-based partitioning, [86](#)
 - hash-based sharding, [86–87](#)
 - hashing-based approaches, [125](#)
 - effect of hash table size, [136–137](#)
 - table, [123, 128–129](#)
 - Hash() function, [129–130](#)
 - HBase (Apache), [18, 45, 64, 163, 164, 168, 169, 174, 181, 186, 201, 203–205, 279, 282, 289](#)
 - HCatalog, [205](#)
 - HCI, *see* [Human–computer interface](#)
 - HDDs, *see* [Hard disk drives](#)
 - HDFS, *see* [Hadoop distributed file system](#)
 - Hector, [433](#)
 - Hierarchical clustering algorithms, [217](#)
 - Hierarchical data objects, [90](#)
 - Hierarchical DBMS, [149](#)
 - High-level language, [151](#)
 - High-performance computing (HPC), [161](#)
 - High data velocity, [154](#)
 - HIHO, *see* [Hadoop In Hadoop Out](#)
 - Hive-Web-Interface (HWI), [187](#)
 - Hive (Apache), [64, 163, 164, 170, 172, 174, 181, 182, 185–187, 195, 203–205](#)
 - HiveQL, [164](#)
 - queries, [203](#)
 - Horizontal fragmentation, [5](#)
 - Horizontal scaling, [49, 92–93](#)
 - “Hot” analytics, [311–312](#)
 - HPC, *see* [High-performance computing](#)
 - HPDL, *see* [Hadoop Process Definition Language](#)
 - HQL, *see* [Hypertable Query Language](#)
 - HTTP API, querying data using, [341–353](#)
 - HUE, *see* [Hadoop user environment](#)
 - Hue (Apache), [188, 209](#)
 - editor, [205](#)
 - Huffman algorithm, [110](#)
 - Huffman encoding, [110](#)
 - Human–computer interface (HCI), [194](#)
 - Human Genome Project, [144, 147, 171](#)
 - HWI, *see* [Hive-Web-Interface](#)
 - Hybrid algorithm, [100](#)
 - Hybrid Cloud, [194](#)
 - Hybrid fragmentation, [5](#)
 - HyperTable, [279, 282–283](#)
 - Hypertable Query Language (HQL), [282](#)
- ## I
- IAuthority interface, [264](#)
 - IBM Cloudbant, [333](#)
 - NoSQL database capabilities, [333–340](#)
 - IBM PureData System for Analytics, [418](#)
 - IBM SoftLayer Cloud, [426](#)
 - ICA, *see* [Independent component analysis](#)
 - ICT-based systems, [291](#)
 - Identifiers, *see* [Generation of Unique IDs \(GUIDs\)](#)
 - IDF, *see* [Inverse Document Frequency](#)
 - IDL, *see* [Interface Description Language](#)
 - IDS alert correlation and sensor placement, [296](#)
 - ILM, *see* [Information Lifecycle Management](#)
 - Immunity requirement, [277](#)
 - Inconsistencies, [22](#)
 - Incremental imports, [57–58, 63, 69](#)
 - Incremental MapReduce, [333](#)
 - Incremental replication, [94–96](#)
 - Independent component analysis (ICA), [216](#)
 - Indexes, [391](#)
 - InfluxDB, [341](#)
 - Information extraction threat, [244](#)
 - Information Lifecycle Management (ILM), [179](#)
 - Information patterns, [391–392](#)

Information Visualization (IVIS), 189
 Inheritance, 238–239
 “Injection Attacks”, 244
 Input validation, 266
 Instance Pooling, 26
 Integration process, 273
 Interface Description Language (IDL), 29
 Internal constraints, 21
 Internet of Things (IoTs), 45, 159
 Internode communication, 259, 277
 Intrusion detection system, 296
 Inverse Document Frequency (IDF), 122
 IoTs, *see* [Internet of Things](#)
 Isolation, 17
 Iterative clustering module, 225
 design on RHadoop, 226–228
 IVIS, *see* [Information Visualization](#)
 IVIS4BigData reference model, Big Data
 analysis infrastructures based on,
 190–192

J

J2EE transaction
 in EJB, 32, 33–34
 transaction in MTS, 36–38
 JAQL, 163, 164, 169
 Java
 application program interface using,
 374–375
 installation, 58–59
 Java script, application program interface
 using, 372–374
 KV access, 410
 table access, 410–412
 Java Database Connectivity (JDBC), 187
 connection using connection pool, 28
 Java Driver, 410
 Java KV access, 410
 Java table access, 410–412
 Java Enterprise Edition (JEE), 423
 Java Naming and Directory Interface (JNDI),
 33
 JavaScript Object Notation (JSON), 155, 164
 database, 365
 data store, 333
 Java Transaction API (JTA), 31
 transaction, 37
 Java Transaction Service (JTS), 31
 Java Virtual Machine (JVM), 183, 201–202, 220
 JDBC, *see* [Java Database Connectivity](#)
 JEE, *see* [Java Enterprise Edition](#)
 JITA, *see* [Just In Time Activation](#)

JNDI, *see* [Java Naming and Directory Interface](#)
 Job scheduler, 95
 JobTracker, 220
 JSON, *see* [JavaScript Object Notation](#)
 JTA, *see* [Java Transaction API](#)
 JTS, *see* [Java Transaction Service](#)
 Just In Time Activation (JITA), 31
 JVM, *see* [Java Virtual Machine](#)

K

Kafka (Apache), 163, 167, 184–185
 Kafka (Apache), 184
 KB-DSS, *see* [Knowledge-based decision
 support systems](#)
 KEGG, *see* [Kyoto Encyclopedia of Genes and
 Genomes](#)
 Kerberos-like high-level protocols, 244
 Key database security functions, 274–275
 Key-value
 database, 168, 355, 421–422
 NoSQL database, 254, 280
 pair database, 18, 19
 store, 406
 K-means++ clustering algorithm, 216, 221–222
 contributions to, 224–225
 implementation for, 225
 improvisations of, 221
 parallel, 222–223, 226
 research challenges, 223–224
 serial K-means++ procedure, 226
 K-nearest neighbor technique, 147
 Knowledge-based decision support systems
 (KB-DSS), 194
 KVLite, 406
 kvroot, 407
 KVStore, 405–406
 KVStore.delete() method, 410
 KVStore.put() method, 410
 Kyoto Encyclopedia of Genes and Genomes
 (KEGG), 148

L

Languages in DBMS, 150–152
 LANs, *see* [Local area networks](#)
 Large-scale computing infrastructures, 120
 LDAP, *see* [Lightweight Directory Access
 Protocol](#)
 Levenshtein distance, 122
 License (Apache), 188, 312
 Lightweight Directory Access Protocol (LDAP),
 240

- Linkage points, identifying, 124
 - LinkedIn Voldemort*, 168
 - Linking records, 128
 - Linux operating system, 59, 135, 313, 314, 355, 365
 - List scheduling, 101
 - first-come-first-serve algorithm, 101–102
 - max–min algorithm, 102
 - min–min algorithm, 102
 - round-robin algorithm, 102
 - LM, *see* [Lock Manager](#)
 - Local area networks (LANs), 2–3
 - Local consistency, 94
 - Local databases, 91
 - Local transaction, 37
 - Lock-based approach, signature set selection, 132–133
 - Lock-free signature selection, 133–134
 - Lock Manager (LM), 7
 - Log data, 201
 - Logical attack graph, 294
 - Logic programming-based approach, 294
 - Low-level language, 151
- M**
- Machine learning (ML), 146, 209, 218
 - JSON, 164
 - library of Spark, 181, 186
 - technique, 209, 221, 248
 - tool, 174
 - tools and techniques, 187–188
 - Macromolecules, 145
 - Mahout (Apache), 174, 178, 186, 187, 200, 209
 - Map Function, 228, 229
 - MapReduce, 105–106, 160, 202–203, 208, 209, 225, 244
 - Apache, 165, 180–181
 - Apache Tez, 208
 - automated, high-throughput exploration of PSP relationship, 173–174
 - for BWT/SA construction, 172
 - Hadoop, 161, 169
 - in HDFS, 221
 - Hive queries, 203
 - implementation, 161
 - incremental, 333
 - module for computing cost function, 226
 - parallel genome indexing with, 171–172
 - Pig Latin, 202–203
 - programming model, 218–219, 221
 - PSP relationship, 172–173
 - serial K-means++ and parallel K-means++ on, 229
 - task flow, 219
 - YARN, 164–165
 - Markov decision process model (MDP model), 283
 - Masking technique, 263
 - Massachusetts General Hospital Utility Multi-Programming System (MUMPS), 155
 - Massively Parallel Processing (MPP), 104
 - Master server, 204
 - Master–slave
 - asynchronous replication, 355
 - replication, 16
 - MATCH clause, 384–385, 388, 391
 - Material science research, 172
 - automated, high-throughput exploration of PSP relationship using MapReduce, 173–174
 - Max–Min algorithm, 102
 - MBMS, *see* [Model base management system](#)
 - McKinsey Global Institute (MGI), 417
 - MD5Hash, 240
 - MDP model, *see* [Markov decision process model](#)
 - Medicine, Hadoop tools applications, 171–172
 - MemcacheDB, 122
 - Memory requirement, 138
 - MERGE query, 385, 390
 - Message-Passing Interface (MPI), 100, 104–105
 - MGI, *see* [McKinsey Global Institute](#); [Mouse Genome Informatics](#)
 - Microsoft Transaction Server (MTS), 25
 - comparison with EJB and, 38
 - distributed transaction management in J2EE, 31
 - distributed transaction support in, 31
 - transactional model in MTS, 34–38
 - Minimum cost network hardening
 - approach, 296
 - Min–Min algorithm, 102
 - Mixed fragmentation, 5
 - ML, *see* [Machine learning](#)
 - MLib, 208–209
 - MLlib (Apache), 179, 181, 188
 - Model-based security-aware elasticity, 283
 - Model-checking techniques, 294
 - Model base management system (MBMS), 194
 - Mongify, 52; *see also* [Apache Sqoop](#); [Neo4J](#)
 - data migration using, 52
 - importing data from MySQL using, 52–55
 - importing specific rows or columns, 55–57
 - incremental imports, 57–58
 - installation and configuration, 52

- open source, 52
 - open source tool, 58
 - MongoDB, 83, 265, 266, 279–280, 401;
 - see also CouchDB; Graph database; RethinkDB
 - advantages, 267–268
 - architecture, 87–88
 - commands, 402–403
 - consistency, 88–89
 - creating user and authentication, 241–242
 - database, 273
 - data model, 83–85
 - downloading and installing, 401–402
 - importing specific rows or columns from MySQL, 55–57
 - incremental imports, 57–58
 - open source tool, 58
 - RDBMS table, 266–267
 - replication and failure handling, 89
 - role-based access control, 241, 242–243
 - security architecture, 264, 268
 - security measures in, 269
 - security weaknesses, 263–265
 - sharding, 85–87
 - SQL to, 52
 - terminologies, 402
 - Mongo Query Language, 279
 - Monotonic
 - reads, 88–89
 - writes, 89
 - Mouse Genome Informatics (MGI), 147
 - MPI, see Message-Passing Interface
 - MPP, see Massively Parallel Processing
 - MTS, see Microsoft Transaction Server
 - MTS model, 30–31
 - Multiple-prerequisite attack graph, 294
 - Multiple processors, 3
 - Multistage Vulnerability AnaLysis (MulVAL), 294
 - MultiUtility Tool, 356
 - MulVAL, see Multistage Vulnerability AnaLysis
 - MUMPS, see Massachusetts General Hospital Utility Multi-Programming System
 - MySQL, 17; see also Not only SQL (NoSQL)
 - using Apache Sqoop, importing data from, 60–61
 - data mapping to Java or Hive data types, 63–64
 - by exporting SQL data, 58
 - Free-Form Query Imports and Incremental Imports, 63
 - to Hadoop, data migration from, 58
 - importing data using Apache Sqoop, 60–61
 - importing specific rows or columns, 55–57, 62–63
 - installing and configuring Apache Sqoop, 58–59
 - using Mongify, importing data from, 52–55
- ## N
- Name Node software, 219–220
 - Nasa1K+1K data set, 137, 138
 - Native graph,
 - databases, 298
 - processing, 298
 - storage, 298, 379
 - Natural Language Processing (NLP), 195
 - NCP security metric, see Network compromise percentage security metric
 - Nearest neighbor method, 147
 - Neo4J, 64, 292, 299–300; see also Apache Sqoop; Mongify
 - Cypher clauses, 385
 - Cypher Query Language, 69, 383–384
 - data types and data access mechanisms, 383
 - Delete command, 389
 - ER model for social contacts application, 386
 - graph database, 38 2
 - graph modeling for social contacts application, 386–388
 - high-level architecture, 382
 - importing data from MySQL using, 64–68
 - importing rows or columns, 68–69
 - incremental imports, 69
 - installation and configuration, 64
 - installing, 397–398
 - loading large amounts of data from CSV, 389
 - Match clauses, 384–385
 - open-source NoSQL database, 64
 - relational data model, 385–386, 387
 - Social Contacts graph creation, 389–392
 - testing model, 388–389
 - use cases, 69, 392–397
 - Neo Technology, 382, 424
 - Nested transactions, 12–13
 - NetSPA, see Network Security Planning Architecture
 - Network CCS, see Network concurrency control scheduler
 - Network compromise percentage security metric (NCP security metric), 296–297

- Network concurrency control scheduler (Network CCS), 10
 - Network DBMS, 149, 150
 - Network forensics analysis, 296
 - Network Security Planning Architecture (NetSPA), 294
 - New-generation database systems, 416
 - NewSQL, 170
 - Next generation sequencing, 155
 - NFRs, *see* [Nonfunctional requirements](#)
 - NLP, *see* [Natural Language Processing](#)
 - NNZ counts, *see* [Number of nonzero counts](#)
 - Nodes, *see* [Entities](#)
 - NodeTool, 330–331
 - Nonfunctional requirements (NFRs), 421
 - Nonprocedural language, 151
 - Non-Redundant DataBase (NRDB), 147
 - Nonrelational database, 157, 204; *see also*
 - [Relational database management system \(RDBMS\)](#)
 - ACID property and, 17–18
 - distributed transaction processing in, 15–16
 - Nonrelational data model, 99–100; *see also*
 - [Relational data model](#)
 - Nonsequence-centric database, 146–147
 - Non-SQL, *see* [Not only SQL \(NoSQL\)](#)
 - Normalized data models, 83, 84
 - Not only SQL (NoSQL), 14, 15, 17, 99, 100, 143, 152, 153, 168–169
 - architectural issues, 276–277
 - architectural security for, 275
 - auditing process, 262
 - authentication, 257–258
 - authorization, 261–262
 - Big Data and NoSQL in biological research, 154–157
 - biological database, 145–148
 - biological DBMS, 148–154
 - Cassandra NoSQL database, 424–426
 - cloud databases, 420–421
 - columnar databases, 422–423
 - comparison of relational database security and, 255–256
 - comparison results, 288–289
 - CouchDb, 90–96
 - data-centric approach to security, 262–263
 - databases, 75, 88, 114–116, 122, 209, 271, 273–274, 311–312, 323, 333, 419, 420
 - database security, 272
 - data stores, 241
 - DB for repository design for IoT, 45
 - distributed environment, 256–257
 - in distributed transaction processing, 18–21
 - document databases, 423
 - EJB implementation in, 27
 - encryption, 262
 - for Enterprise, 423–424
 - environmental and process control, 262
 - fine-grained authorization and access control, 258
 - Google’s BigTable, 80–83
 - graph databases, 421
 - HDFS architecture, 276
 - IBM Cloudant NoSQL database capabilities, 333–340
 - implementation, 157
 - information extraction threat, 244
 - integration using Hadoop, 29–30
 - key database security functions, 274–275
 - key–value databases, 421–422
 - MongoDB, 83–89
 - NoSQL cluster security, 260
 - performance management in, 114
 - privacy, 248, 260, 285–288
 - protection of data, 258–259
 - relational databases, 252–253
 - role-based access control for MongoDB, 241–243
 - safeguarding integrity, 258
 - securing NoSQL applications, 265–266
 - security, 252
 - security and privacy solutions for data stores, 244–247
 - security issues of databases, 277–284
 - security reference architecture, 260, 261
 - security solutions of traditional DBMS, 237–241
 - security threats, 253–255, 256
 - security weakness in, 266–269
 - SimpleDB, 76–80
 - SQL injection attack, 244
 - system model of, 241
 - third-party tools *vs.* database vendors, 277
 - types of, 421
 - user access management, 260
 - workflow from data to knowledge, 144
 - NRDB, *see* [Non-Redundant DataBase](#)
 - n-tier application model, 33
 - Number of nonzero counts (NNZ counts), 230
- O**
- Object identifier (OID), 152
 - Object-oriented DBMS (OODBMS), 149–150, 151–152
 - Object-oriented programming (OOP), 151

- ODBC, *see* [Open Database Connectivity](#)
- OEP, *see* [Oozie Eclipse Plugin](#)
- OID, *see* [Object identifier](#)
- OLAP, *see* [Online Analytical Processing](#)
- OLTP, *see* [Online transaction processing](#)
- OMIM, *see* [Online Mendelian Inheritance in Man](#)
- On-demand self-service, [193–194](#)
- Online Analytical Processing (OLAP), [203](#)
- Online Mendelian Inheritance in Man (OMIM), [148](#)
- Online transaction processing (OLTP), [203](#), [426](#)
- OODBMS, *see* [Object-oriented DBMS](#)
- OOOP, *see* [Object-oriented programming](#)
- Oozie (Apache), [163](#), [182](#), [200](#), [204–205](#), [206](#), [209](#)
 - bundle, [205](#)
 - coordinator jobs, [205](#)
 - workflow jobs, [205](#)
- Oozie Eclipse Plugin (OEP), [205](#)
- Opaque data objects, [90](#)
- Open-source front-edge database, [311–312](#)
- Open Database Connectivity (ODBC), [187](#)
- OPT, *see* [Distributed Optimistic](#)
- Optimal infrastructures for Big data analytics, [418–420](#)
- Oracle NoSQL database, [405](#); *see also* [Apache Cassandra database](#)
 - architecture, [405](#)
 - installation, [407](#)
 - Java Driver, [410–412](#)
 - key–value store, [406](#)
 - KVLite, [406](#)
 - loading data from CSV, [412–413](#)
 - read and write operations, [406](#)
 - starting of database, [407–408](#)
 - tables, [408–409](#)
- OS X, [355](#), [365](#)
- P**
- P2P model, *see* [Peer-to-peer model](#)
- Parallel algorithms
 - lock-free signature selection, [133–134](#)
 - for signature selection on GPGPU, [132](#)
 - signature set selection using lock-based approach, [132–133](#)
- Parallel DBMS and DDBMS, [3–4](#)
- Parallel Genome Indexing with MapReduce, [171–172](#)
- Parallelization, [131](#)
- Parallel K-means++ clustering algorithm, [222](#)
 - algorithm for, [223](#)
 - initial centroid selection for, [226](#)
 - initialization method, [225](#), [226](#)
 - on MapReduce paradigm, [229](#)
 - size-up analysis for distributed, [231](#)
- Parallel processing-based approaches, [125](#)
- Parallel Virtual Machine (PVM), [104](#)
- Partition key, [426](#)
- Partition tolerance, [17](#)
- PCA, *see* [Principal component analysis](#)
- PDB, *see* [Protein Data Bank](#)
- Peers, CouchDB architecture, [90](#)
- Peer-to-peer model (P2P model), [263](#)
 - distributed system, [209–210](#)
 - replication, [16](#)
- Performance evaluation, clustering algorithm, [230](#)
 - clusters evaluations and determining optimal number of clusters, [233–234](#)
 - datasets, [230–231](#)
 - size-up analysis, [231](#)
 - total time taken and number of iterations required for convergence, [231–233](#)
 - visualization results, [234–235](#)
- Persistence znode, [207](#)
- PGRR, *see* [Pittsburgh Genome Resource Repository](#)
- Pig (Apache), [163](#), [164](#), [179](#), [182](#), [186](#), [202–203](#)
- Pig Latin, [202–203](#)
- Pipelining data across storage systems, Google
 - cloud dataflow for, [69–70](#)
- Pittsburgh Genome Resource Repository (PGRR), [156](#)
- PL/SQL program, [15](#)
- Portable Operating System Interface for Unix (POSIX), [355](#)
- PostgreSQL database system, [237–238](#); *see also* [Not only SQL databases \(NoSQL databases\)](#)
 - PostgreSQL user authentication, [240–241](#)
 - RBAC for, [239–240](#)
- PQL, *see* [Program Query Language](#)
- Primary copy 2PL, [7](#)
- Primary databases, [147](#)
- Primary key, [52](#)
- Primary member, [89](#)
- Principal component analysis (PCA), [216](#), [224](#), [229](#)
- Privacy, [248](#)
 - ABAC model, [246](#)
 - CA-RBAC, [246–247](#)
 - challenges of NoSQL databases, [285](#)
 - content-based access control, [245](#)
 - of data owners and query users, [286–287](#)
 - existing privacy problems solution, [287–288](#)
 - FGAC, [246](#)

- Privacy (*Continued*)
 - preserving solutions, 248
 - R-ABAC model, 247
 - solutions for NoSQL data stores, 244
 - spatial and location-based access model, 245
 - TRBAC model, 244–245
 - of user data, 260
 - violation under NoSQL data stores, 248
 - Private Cloud, 194
 - Procedural language, 151
 - Process–Structure–Property relationship (PSP relationship), 172–173
 - automated, high-throughput exploration of, 173–174
 - Process-to-Structure modeling, 173
 - Programmer-friendly packages, 432
 - Programming paradigm for cloud applications, 103
 - Map Reduce, 105–106
 - MPI, 104–105
 - PVM, 104
 - Program Query Language (PQL), 299
 - Protein Data Bank (PDB), 148
 - Proteomic research, 156
 - PSP relationship, *see* Process–Structure–Property relationship
 - Public Cloud, 194
 - Put operation, 83
 - PVM, *see* Parallel Virtual Machine
- Q**
- Quality of Service (QoS), 100, 421
 - Query
 - Model, 76
 - querying data using HTTP API, 341–353
 - routers, 85
- R**
- R-ABAC model, *see* Role and attribute-based access model
 - Range-based
 - partitioning, 86
 - sharding, 86
 - Ranger (Apache), 207–208
 - Ranger, 207
 - plugins, 207–208
 - portal, 207–208
 - Ravel Law, 397
 - RBA approach, *see* Risk-based audit approach
 - RBAC model, *see* Role-based access control model
 - RDBMS, *see* Relational database management system
 - RDD, *see* Resilient distributed datasets
 - RDF, *see* Resource description framework
 - Read and write operations, 406
 - Read one, write all (ROWA), 7
 - Read request, 9, 209–210
 - Real-time
 - analytics, 418
 - protection, 275
 - REAs, *see* Richer Enterprise Applications
 - Recommendation engines, 393
 - Record-wise signature set, 126
 - Record linkage, 120, 124
 - approach, 135
 - data encoding and DF update, 127
 - forming signature sets, 127
 - using GPGPU, 125
 - identifying candidate records, 127–128
 - linking records, 128
 - preprocessing, 126
 - Red Hat variants, 313
 - Redis, 355
 - advantages of, 356
 - Browse button to placing program’s shortcut, 358
 - example, 363
 - installation process, 359
 - License Agreement screen, 357
 - Redis server, 360–362
 - Red teams, 293
 - Reduce Function, 228, 229
 - References, 83
 - Relational database management system (RDBMS), 14, 17, 48–49, 150, 151, 152, 200–201, 203, 237
 - hitting glass ceiling of data management limits, 49
 - RDBMS-based approach, 300
 - Relational databases, 87, 99, 252–253, 272, 379, 381, 422
 - comparison of NoSQL security and, 255–256
 - distributed transaction processing in, 14–15
 - Relational data model, 298, 387, 390
 - for social contacts application, 385–386
 - SQL, 292
 - Relational model, 291
 - Reliable data center services, 106
 - Remote databases, 91
 - Remote Procedure Call protocol (RPC protocol), 432
 - REPL commands, 330

- Replication, 16
 - CouchDb, 96
 - factor, 406
 - Google's BigTable, 82
 - incremental, 94–96
 - MongoDB, 89
 - nodes, 406
 - reliable storage, 82
 - Replica Placement strategy, 211
 - sequential storage, 83
 - sets, 89
 - strategy, 5
 - structured storage, 83
 - Resilient distributed datasets (RDD), 165, 188, 208–209
 - Resource description framework (RDF), 195, 299
 - Resource Manager (RM), 35, 36, 37–38, 39
 - Resource Object, 32
 - REST APIs, 79, 183, 207
 - RethinkDB, 365; *see also* MongoDB
 - application program interface, 372–375
 - Dashboard, 367
 - database creation, 368–369
 - deletion of row from table, 372
 - installation process of, 366
 - network option, 367
 - selection of data from table, 371
 - single data insertion, 370
 - tables, 368
 - RETURN clause, 391
 - RHadoop platform, 218, 221; *see also* Hadoop Ecosystems
 - cluster assignment module on RHadoop, 228, 229
 - iterative clustering module design on, 226–228
 - Richer Enterprise Applications (REAs), 423
 - Risk-based audit approach (RBA approach), 264
 - R language, 221
 - RM, *see* Resource Manager
 - Role-based access control model (RBAC model), 237, 243, 253, 258; *see also* Temporal role-based access control model (TRBAC model)
 - CA-RBAC, 246–247
 - constrained RBAC, 239
 - for database systems, 238–239
 - flat RBAC, 238
 - health service organization, 246
 - model with inheritance, 238
 - MongoDB, 241–243
 - PostgreSQL, 239–240
 - Role and attribute-based access model (R-ABAC model), 247
 - Round-robin algorithm, 102
 - Row-oriented database management system, 422
 - ROWA, *see* Read one, write all
 - Row key, *see* Partition key
 - Rows
 - Data Model of BigTable, 81
 - importing, 62–63, 68–69
 - from MySQL, 55–57
 - RPC protocol, *see* Remote Procedure Call protocol
 - RTC, 246–247
 - Ruby, application program interface using, 374
 - RubyGems software, 52; *see also* Mongify Rule induction, 147
- S**
- SA, *see* Suffix array
 - Safeguarding integrity, 258
 - SAP HANA, 418
 - SASL, *see* Simple Authentication and Security Layer
 - Scalability, 132
 - Scalable record linkage technique
 - blocking mechanism, 124–125
 - cloud computing-based approaches, 125
 - cloud computing, 120
 - data sets, 135
 - DF, 122
 - document-oriented databases, 121
 - existing methods, 124
 - experimental methodology, 134
 - experimental platform, 135
 - F-measure vs. memory requirement, 138
 - GPGPUs, 123
 - hashing-based approaches, 125
 - hashing table, 123
 - effect of hash table size, 136–137
 - identifying linkage points, 124
 - Levenshtein distance, 122
 - metrics, 135
 - NoSQL databases, 122
 - parallel algorithms for signature selection on GPGPU, 132–134
 - parallel processing-based approaches, 125
 - performance comparison, 138–139
 - performance of hash-based approach for data encoding, 136
 - performance of signature selector-based data pruning, 137

- Scalable record linkage technique (*Continued*)
 - record linkage using GPGPU, 125–128
 - research directions, 139
 - result, 135
 - signature selection, 128–132
 - similarity metric, 122
 - similarity techniques optimizations, 125
 - stage optimizations, 124
 - tree data model, 123–124
- Scaling out, *see* Horizontal scaling
- Scaling up, *see* Vertical scaling
- Scan operation, 83
- Scatter plot tool, 229, 234
- Scheduler, 6
 - Apache Oozie Workflow Scheduler, 163
 - CCS, 10
 - job scheduler, 95
 - task, 124
- SDK, *see* Software development kit
- Secondary databases, 147
- Secondary members, 89
- Secure computation, 272
- Secure data transactions, 272
- Secure Shell protocol, 244
- Secure Sockets Layer protocol (SSL protocol), 240
- Security
 - ABAC model, 246
 - CA-RBAC, 246–247
 - Cassandra database, 280–281
 - condition nodes, 295
 - constraint, 21
 - content-based access control, 245
 - CouchDB, 281–282
 - in distributed transaction processing systems, 21–22
 - FGAC, 246
 - HBase, 282
 - HyperTable, 282–283
 - issues of NoSQL databases, 277
 - mechanisms, 278
 - model-based security-aware elasticity, 283
 - model, 278
 - MongoDB, 279–280
 - in NoSQL databases, 279
 - objectives, 278
 - policy, 278
 - problems of NoSQL environments, 283–284
 - R-ABAC model, 247
 - security requirements, 277–278
 - solutions for NoSQL data stores, 244
 - solutions of traditional database management systems, 237–241
 - spatial and location-based access model, 245
 - threats in NoSQL databases, 253–255, 256–260
 - TRBAC model, 244–245
 - weaknesses of Cassandra and MongoDB, 263–265
 - weakness in typical NoSQL database, 266–269
- Seeking, 83
- Selective replication, 6
- SensiDB, 170
- Separation of duties (SOD), 239
- SEQUEL, *see* Structured English QUery Language
- Sequential znode, 207
- Serial K-means++ clustering algorithms, 218, 225–226
 - drawback of, 222–223
 - initial centroid selection for, 226
 - on MapReduce paradigm, 229
 - size-up analysis for distributed, 231
- Service-oriented computing, 100
- SET query, 385
- Sharding, 16
 - CouchDb, 92–93
 - data partitioning, 85
 - Google’s BigTable, 82
 - hash-based sharding, 86–87
 - by MongoDB, 85
 - range-based sharding, 86
 - scaling out, 92–93
 - shard keys, 85–86
- Shard keys, 85–86
- “Shards”, 82, 85
- Shared nothing cluster, 94
- Shell commands, 211
- Shifts, 91
- ShiftServer, 90
- ShiftSpace, 90, 91
- SignatureSel data structure, *see* Signature Selector data structure
- Signature selection, 128
 - baseline hashing-based approach, 128–129
 - collision management, 132
 - forming signature sets, 130–131
 - parallel algorithms for signature selection on GPGPU, 132–134
 - parallelization, 131
 - scalability, 132
 - SignatureSel, 129–131
- Signature selector-based data pruning, 137
- Signature Selector data structure (SignatureSel data structure), 129–130

- update signature selector, 131
- Signature set(s)
 - combine phase, 133
 - extract phase, 132–133
 - forming, 130–131
 - selection using lock-based approach, 132
- Simple Authentication and Security Layer (SASL), 282
- SimpleDB, 76
 - consistency, 79–80
 - data model, 77–79
- Simple Object Access Protocol (SOAP), 29, 272
- Single-instruction multiple-thread machines (SIMT machines), 121
- Single Program Multiple Data (SPMD), 104
- “Sink”, 183, 184
- SM, *see* Streaming multiprocessor
- Smart clustering algorithms, 224
- SMaSh framework, 124
- Snapchat, 288
- SOAP, *see* Simple Object Access Protocol
- Social computing sites, 423
- Social Contacts
 - finding information patterns, 391–392
 - graph creation, 389
 - graph modeling for, 386–388
 - indexes and constrains, 391
 - loading books from CSV, 389–390
 - querying database, 390–391
 - relational data model for application, 385–386
- Social network, 381, 393
 - analysis, 378
 - graph, 380
- SOD, *see* Separation of duties
- SoftLayer, 426
- Software development kit (SDK), 312
- Solid state drives (SSDs), 166
- Spark (Apache), 165, 187–180, 208–209
 - machine learning library, 181, 186
 - Spark Core, 208–209
- SPARQL, 299
- Sparse hashtable, 83
- Spatial and location-based access model (Geo-RBAC model), 245, 246
- SP cores, *see* Streaming processor cores
- SPMD, *see* Single Program Multiple Data
- SQL, *see* Structured query language
- SQL to MongoDB
 - importing data from MySQL using Mongify, 52–55
 - importing specific rows or columns from MySQL, 55–57
 - incremental imports, 57–58
 - installation and configuration Mongify, 52
 - Mongify, open source, 52
 - open source tool, 58
- SQL to Neo4J, 64
 - Cypher Query Language, 69
 - importing data from MySQL using Neo4J, 64–68
 - importing specific rows or columns, 68–69
 - incremental imports, 69
 - installing and configuring Neo4J, 64
 - use case for joining two tables, 69
- Sqoop (Apache), 200
- Sqoop 2 (Apache), 182, 183
- SSDs, *see* Solid state drives
- SSE criterion, *see* Sum of squared error criterion
- SSL protocol, *see* Secure Sockets Layer protocol
- Standard columns, 425
- State-of-the-art algorithm, 160
- State enumeration graph, 293, 294
- Static column family, 425
- Static scheduling, 101
- Storage layer, 108–109
- Storage tiering, 108–109
- Storage Virtualization, 108–109
- Storm (Apache), 165, 167
- Stream
 - computing, 418
 - data, 201
- Streaming multiprocessor (SM), 123
- Streaming processor cores (SP cores), 123
- String edit distance, *see* Levenshtein distance
- Strings, 83
- Structured English QUERy Language (SEQUEL), 151
- Structured query language (SQL), 14, 151, 203, 298; *see also* Not only SQL (NoSQL)
 - comparison of NoSQL and, 21
 - databases, 416
 - to Hadoop and Hadoop to SQL, 200
 - injection attack, 244
 - SQL-driven RDBMS, problems with, 51
 - SQL-like queries, 203, 341
 - SQL-on-Hadoop, 170
 - SQL-type query language, 164
- Suffix array (SA), 171
- Summation, 219
- Summingbird, 165
- Sum of squared error criterion (SSE criterion), 229, 233–234
- Support cloud computing services, *see* Large-scale computing infrastructures

Swissprot, record linkage process, 136–137
 Symmetric model, 239
 Synchronization Object, 32
 Synchronous transactions, 23
 System model of NoSQL data stores, 241
 system.time(), 231

T

Tablet, 81
 TAIR, *see* [The Arabidopsis Information Resource](#)
 Task scheduler, 124
 Task scheduling in cloud, 101
 DAG scheduling, 102–103
 list scheduling, 101–102
 TCGA, *see* [The Cancer Genome Atlas](#)
 TDE technique, *see* [Transparent data encryption technique](#)
 Temporal role-based access control model (TRBAC model), 244–245
 Terminator Object, 32
 Texture memory (TM), 123
 Tez (Apache), 208
 The Arabidopsis Information Resource (TAIR), 147
 The Cancer Genome Atlas (TCGA), 156
 Thin provisioning, 108, 111
 Third-party tools, 277
 Thread-Per-Client model, 284
 Thrift protocol, 432
 Timestamps, 81–82
 Time to live (TTL), 426
 TITAN Distributed Graph Database (TitanDB), 169
 TLS protocol, *see* [Transport Layer Security protocol](#)
 TL transaction, *see* [Top-level transaction](#)
 TM, *see* [Texture memory](#); [Transaction manager](#)
 Tokenization security, 262–263
 Top-level transaction (TL transaction), 12
 Topological vulnerability analysis (TVA), 294
 Topologies, 165
 TPS, *see* [Transaction processing system](#)
 Traditional database management systems;
 see also [Relational database management system \(RDBMS\)](#)
 PostgreSQL user authentication, 240–241
 RBAC, 237–240
 security solutions of, 237
 Traditional storage provisional model, 111
 Transactional model

 in EJB, 31
 J2EE transaction in EJB, 32, 33–34
 J2EE transaction in MTS, 36–38
 in MTS, 34
 Transaction manager (TM), 7, 10, 23, 35, 37
 Transaction processing system (TPS), 21
 Transaction(s), 10
 client, 10
 data, 200
 management module, 1
 processing models, 11
 transactional server, 10
 Transformation mappings, interaction with, 189–190
 Transparency, 9
 Transparent data encryption technique (TDE technique), 259
 Transport Layer Security protocol (TLS protocol), 240
 TRBAC model, *see* [Temporal role-based access control model](#)
 Tree data model, 123–124
 TSM engine, 341
 TTL, *see* [Time to live](#)
 TVA, *see* [Topological vulnerability analysis](#)
Twitter Summingbird, 165
 Two-phase commit algorithm (2PC algorithm), 24
 2PC protocol, 25
 Two-Phase Locking (2PL), 6
 to DDBMS, 7
 distributed, 7
 primary copy, 7

U

UDF, *see* [User defined functions](#)
 UI, *see* [User interface](#)
 UNION query, 385
 Unix operating system, 59, 93, 182
 URLtable example, 81
 Use case(s), 382, 392
 credit card transactions, 395
 fraud analytics, 394–397
 fraudulent transactions response, 397
 geospatial data, 393–394
 graph search, 394
 recommendation engines, 393
 social network, 393
 User defined functions (UDF), 187, 203, 311–312
 User/feed, 91
 User interface (UI), 163, 207
 User/private, 91

User/public, 91
UserTransaction interface, 33
UTF-8 strings, 282

V

Vagrant cloud virtual machine, 313
Vertical fragmentation, 5
Vertical scaling, 49
Virtualization, 100
 storage, 108–109
 system, 314
Virtual machines (VMs), 101, 283, 314
Visual data views, interaction with, 189
Visualization,
 with Hadoop, 188
 pipeline, interaction with, 189–190
 results, 234–235
 tools, 216, 224, 229
Visual mapping, 190
VMs, *see* Virtual machines
Volume, variety, and velocity (3Vs), 159–160
Vulnerability
 scanners, 275
 scanning tools, 292

W

Walmart, 50
Web-scale applications, 421
Web Container, 33
Web service(s), 27
 back-end database, 293
 EJB integrating, 28–29
 SimpleDB, 76–80
Web Services Description Language (WSDL), 27
WhatsApp, 288

WHERE query, 385, 391
WLCG, *see* Worldwide LHC Computing Grid
Worldwide LHC Computing Grid (WLCG), 273
Wound-Wait (WW), 8
Write operations, 9, 84–85
Write requests, 209–210
WSDL, *see* Web Services Description Language
WW, *see* Wound-Wait

X

XA Specification, 39
 XA interface in DTP, indexed services of,
 38–45
 XA transaction, 37
XML-encoded data set, 126
X/Open Distributed Transaction Processing
 commit phase, 24–25
 commit request phase, 24
 distributed transaction support in EJB and
 MTS, 31–38
 EJB and MTS comparison, 38
 indexed services of XA interface in DTP,
 38–45
 NoSQL DB for repository design for IoT, 45
 technologies in DTP, 25–31
XQuery, 125

Y

Yet Another Resource Negotiator (YARN), 161,
181, 208

Z

Znode, 207
Zookeeper (Apache), 163, 181, 205–207



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>