READING MATERIALS – EDUCATIONAL PURPOSES ONLY

BIG DATA ANALYTICS

TABLE OF CONTENTS

Chapter 1 Introduction to Big Data Analytics 7

1.1 An introduction to Big Data Analytics 7

1.1.1 What is Analytics? 7

1.1.2 What is Big Data Analytics? 8

1.1.3 History and Evolution of Big Data Analytics 9

1.1.4 Values of Big Data Analytics 9

1.1.5 Uses of Big Data analytics across different industries 10

1.2.Big Data Analytics - Overview 11

1.3.Big Data Analytics - Data Life Cycle 11

1.3.1 Traditional Data Mining Life Cycle 11

1.3.2 Big Data Life Cycle 14

1.4.Big Data Analytics - Methodology 17

1.5.ANALYTICS PROCESS MODEL 17

1.6.ANALYTICAL MODEL REQUIREMENTS 19

1.7.1 Types of Data Sources 21

1.7.2 Sampling 22

1.7.3 Types of data elements 23

1.7.4 Visual data exploration and exploratory statistical analysis 24

1.7.5 Missing values 25

1.7.6 Outlier detection and treatment 26

1.7.7 Categorization 29

1.7.8 Variable selection 33

1.7.9 Segmentation 35

1.8. TYPES OF DATA 35

1.8.1 Datification: The new forms of data 38

1.8.2 The anatomy of Big Data 46

1.9. Big Data Tools and Techniques 49

1.9.1 Understanding Big Data Storage 49

1.9.2 A Generals Overview of High-Performance Architecture 49

1.9.3 HDFS 50
1.9.4 MapReduce and YARN 52
1.9.5 Expanding the Big Data Application Ecosystem 53
1.9.6 ZOOKEEPER 54
1.9.7 HBASE 54
1.9.8 HIVE 54
1.9.9 PIG 55
1.9.10 MAHOUT 56
1.9.11 Considerations 57

Chapter 2 Big Data Platforms - Hadoop 58
2.1 A Brief History of Hadoop 58
2.2 Big Data & Hadoop – Restaurant Analogy 59
2.3 What is Hadoop? 64
2.3.1 Hadoop-as-a-Solution 64
2.3.2 Hadoop Features 65
2.3.3 Hadoop Core Components 66
2.3.4 Hadoop Ecosystem 68
2.3.5 Examples of Hadoop - 5 Real-World Use Cases 69
2.3.6 Last.fm Case Study 70
2.3.7 Advantages of using Hadoop 71
2.3.8 Disadvantages of using Hadoop 72
2.3.9 Is Hadoop an efficient use of resources? 73
2.3.10 The business case for Hadoop 74
2.3.11 Another way to look at value 74
2.3.12 What does Hadoop replace? 75
2.3.13 Problems that Hadoop solves 75
2.4 Hadoop Installation 76
2.4.1 Java Installation 76
2.4.2 SSH Installation 77
2.4.3 Hadoop Installation 78

2.5 Hadoop Modules 81
2.5.1 HDFS 81
2.5.2 YARN 87
2.5.3 MapReduce 87

Chapter 3 Big Data Storage and Analytics 93
3.1 Big Data Storage Concepts 93
3.1.1 Clusters 93
3.1.2 File Systems and Distributed File Systems 94
3.1.3 NoSQL 96
3.1.4 Sharding 96
3.1.5 Replication 98
3.1.6 Sharding and Replication 103
3.1.7 CAP Theorem 106
3.1.8 ACID 110
3.1.9 BASE 114
3.1.10 Case Study Example 117
3.2 Apache Mahout 118
3.2.1 Mahout's Story 119
3.2.2 What is Machine Learning? 120
3.2.3 Mahout's Machine Learning Themes 123
3.2.3.1 Recommender engines 123
3.2.3.2 Clustering 131
3.2.3.3 Classification 137
3.2.4 Tackling large scale with Mahout and Hadoop 142

Chapter 4 Machine Learning, Streams and Databse on Spark 144
4.1 Spark: Real Time Cluster Computing Framework 144
4.1.1 Real Time Analytics 144
4.1.2 Why Spark when Hadoop is already there? 146
4.1.3 What is Apache Spark? 147

4.1.4 Features of Apache Spark 148
4.1.5 Getting Started With Spark 150
4.1.6 Using Spark with Hadoop 151
4.1.7 Spark Components 152
4.1.8 Earthquake Detection using Spark 155
4.2 Sentiment Analysis Using Apache Spark 160
4.2.1 What is Streaming? 160
4.2.2 Why Spark Streaming? 160
4.2.3 Spark Streaming Overview 161
4.2.4 Spark Streaming Features 161
4.2.5 Spark Streaming Workflow 161
4.2.6 Spark Streaming Fundamentals 162
4.2.7 Use Case – Twitter Sentiment Analysis 166
4.3 Spark MLlib – Machine Learning Library Of Apache Spark 171
4.3.1 What is Machine Learning? 172
4.3.2 Spark MLlib Overview 173
4.3.3 Spark MLlib Tools 173
4.3.4 MLlib Algorithms 174
4.3.5 Use Case – Movie Recommendation System 176
4.4 Spark SQL Tutorial – Understanding Spark SQL With Examples 181
4.4.1 Why Spark SQL Came Into Picture? 182
4.4.2 Limitations with Hive 182
4.4.3 Spark SQL Overview 182
4.4.4 Spark SQL Libraries 183
4.4.5 Features Of Spark SQL 184
4.4.6 Querying Using Spark SQL 188
4.4.7 Creating Datasets 192
4.4.8 Adding Schema To RDDs 194
4.4.9 RDDs As Relations 197
4.4.10 Caching Tables In-Memory 199
4.4.11 Loading Data Programmatically 199

4.4.12 JSON Datasets 201
4.4.13 Hive Tables 202
References 206

CHAPTER 1

INTRODUCTION TO BIG DATA ANALYTICS

1.1 AN INTRODUCTION TO BIG DATA ANALYTICS [1]

Many of you would have probably heard about Big data Analytics. Have you ever wondered what it is all about and how it can help us? Big data analytics can be defined as a process of examining large and varied data sets. We use advanced analytics techniques against the large data to uncover the hidden patterns, unknown correlations, market trends, customer preferences and other useful information. This helps the organizations to make informed decisions.

To understand Big Data Analytics you have to first understand What Analytics is?

1.1.1 What is Analytics? [1, 3]

Analytics is an encompassing and multidimensional field. It uses mathematics, statistics, predictive modeling and machine-learning techniques to find meaningful patterns and knowledge in recorded data.

Analytics is a term that is often used interchangeably with data science, data mining, knowledge discovery, and others. The distinction between all those is not clear cut. All of these terms essentially refer to extracting useful business patterns or mathematical decision models from a preprocessed data set. Different underlying techniques can be used for this purpose, stemming from a variety of different disciplines, such as:

- Statistics (e.g., linear and logistic regression)
- Machine learning (e.g., decision trees)
- Biology (e.g., neural networks, genetic algorithms, swarm intelligence)
- Kernel methods (e.g., support vector machines)

Basically, a distinction can be made between predictive and descriptive analytics. In predictive analytics, a target variable is typically available, which can either be categorical (e.g., churn or not, fraud or not) or continuous (e.g., customer lifetime value, loss given default). In descriptive analytics, no such target variable is available. Common examples here are association rules, sequence rules, and clustering. Figure 1.1 provides an example of a decision tree in a classification predictive analytics setting for predicting churn.

More than ever before, analytical models steer the strategic risk decisions of companies. For example, in a bank setting, the minimum equity and provisions a financial institution

holds are directly determined by, among other things, credit risk analytics, market risk analytics, operational risk analytics, fraud analytics, and insurance risk analytics. In this setting, analytical model errors directly affect profitability, solvency, shareholder value, the macroeconomic, and society as a whole. Hence, it is of the utmost importance that analytical models are developed in the most optimal way, taking into account various requirements that will be discussed in what follows.

Customer	Age	Recency	Frequency	Monetary	Churn
John	35	5	6	100	Yes
Sophie	18	10	2	150	No
Victor	38	28	8	20	No
Laura	44	12	4	280	Yes

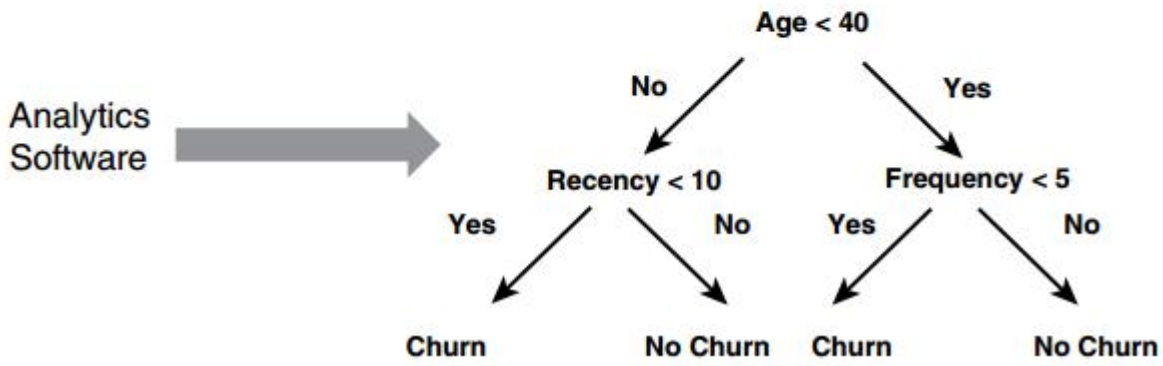


Figure 1.1 Example of Classification Predictive Analytics [3]

1.1.2 What is Big Data Analytics? [1]

As said before Big data analytics examines large amounts of data to uncover hidden patterns, correlations and other insights. With today's technology, it's possible to analyze your data and get answers from it immediately. Big Data Analytics helps you to understand your organization better. With the use of Big data analytics, one can make the informed decisions without blindly relying on guesses.

And it can help answer the following types of questions:

What actually happened?

How or why did it happen?
What's happening now?
What is likely to happen next?

1.1.3 History and Evolution of Big Data Analytics [1]

The concept of big data has been around for years; most organizations now understand that if they capture all the data that streams into their businesses, they can apply analytics and get significant value from it. But even in the 1950s, decades before anyone uttered the term "big data," businesses were using basic analytics essentially numbers in a spreadsheet that were manually examined to uncover insights and trends.

The new benefits that big data analytics brings to the table, however, are speed and efficiency. Whereas a few years ago a business would have gathered information, run analytics and unearthed information that could be used for future decisions, today that business can identify insights for immediate decisions. The ability to work faster – and stay agile – gives organizations a competitive edge they didn't have before.

1.1.4 Values of Big Data Analytics [1]

Figure 1.2 Big Data Analytics

Big data analytics helps organizations harness their data and use it to identify new opportunities. That, in turn, leads to smarter business moves, more efficient operations, higher profits and happier customers. Here are the most important values of Big Data,

1. *Cost reduction*: Big data technologies such as Hadoop and cloud-based analytics bring significant cost advantages when it comes to storing large amounts of data – plus they can identify more efficient ways of doing business.

2. *Faster, better decision making*: With the speed of Hadoop and in-memory analytics, combined with the ability to analyze new sources of data, businesses are able to analyze information immediately – and make decisions based on what they've learned.
3. *New products and services*: With the ability to gauge customer needs and satisfaction through analytics comes the power to give customers what they want. Davenport points out that with big data analytics, more companies are creating new products to meet customers' needs.

1.1.5 Uses of Big Data analytics across different industries[1]

Banking
Large amounts of information will be streaming in into banks, managing all this data and getting proper insights would be possible only with big data analytics. This is important to understand customers and boost their satisfaction, and also to minimize risk and fraud.

Government
When government agencies are able to harness and apply analytics to their big data, they gain significant ground when it comes to managing utilities, running agencies, dealing with traffic congestion or preventing crime.

Health Care
Patient records, Treatment plans, Prescription information. When it comes to health care, everything needs to be done quickly, accurately – and, in some cases, with enough transparency to satisfy stringent industry regulations. When big data is managed effectively, health care providers can uncover hidden insights that improve patient care.

Education
Educators armed with data-driven insight can make a significant impact on school systems, students, and curriculums. By analyzing big data, they can identify at-risk students, make sure students are making adequate progress, and can implement a better system for evaluation and support of teachers and principals.

Manufacturing
Armed with insight that big data can provide, manufacturers can boost quality and output while minimizing waste – processes that are key in today's highly competitive market. More and more manufacturers are working in an analytics-based culture, which means they can solve problems faster and make more agile business decisions.

Retail
Customer relationship building is critical to the retail industry – and the best way to manage that is to manage big data. Retailers need to know the best way to market to customers, the most effective way to handle transactions, and the most strategic way to bring back lapsed business. Big data remains at the heart of all those things.

1.2. BIG DATA ANALYTICS - OVERVIEW [2]

The volume of data that one has to deal has exploded to unimaginable levels in the past decade, and at the same time, the price of data storage has systematically reduced. Private companies and research institutions capture terabytes of data about their users' interactions, business, social media, and also sensors from devices such as mobile phones and automobiles. The challenge of this era is to make sense of this sea of data. This is where big data analytics comes into picture.

Big Data Analytics largely involves collecting data from different sources, munge it in a way that it becomes available to be consumed by analysts and finally deliver data products useful to the organization business.

The process of converting large amounts of unstructured raw data, retrieved from different sources to a data product useful for organizations forms the core of Big Data Analytics.

1.3. BIG DATA ANALYTICS - DATA LIFE CYCLE [2]

1.3.1 Traditional Data Mining Life Cycle [2]

In order to provide a framework to organize the work needed by an organization and deliver clear insights from Big Data, it's useful to think of it as a cycle with different stages. It is by no means linear, meaning all the stages are related with each other. This cycle has superficial similarities with the more traditional data mining cycle as described in CRISP methodology.

CRISP-DM Methodology
The CRISP-DM methodology that stands for Cross Industry Standard Process for Data Mining, is a cycle that describes commonly used approaches that data mining experts use to tackle problems in traditional BI data mining. It is still being used in traditional BI data mining teams.
Take a look at the Fig.1.3. It shows the major stages of the cycle as described by the CRISP-DM methodology and how they are interrelated.

CRISP-DM was conceived in 1996 and the next year, it got underway as a European Union project under the ESPRIT funding initiative. The project was led by five companies: SPSS, Teradata, Daimler AG, NCR Corporation, and OHRA (an insurance company). The project was finally incorporated into SPSS. The methodology is extremely detailed oriented in how a data mining project should be specified.

Let us now learn a little more on each of the stages involved in the CRISP-DM life cycle

Business Understanding − This initial phase focuses on understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a data mining problem definition. A preliminary plan is designed to achieve the objectives. A decision model, especially one built using the Decision Model and Notation standard can be used.

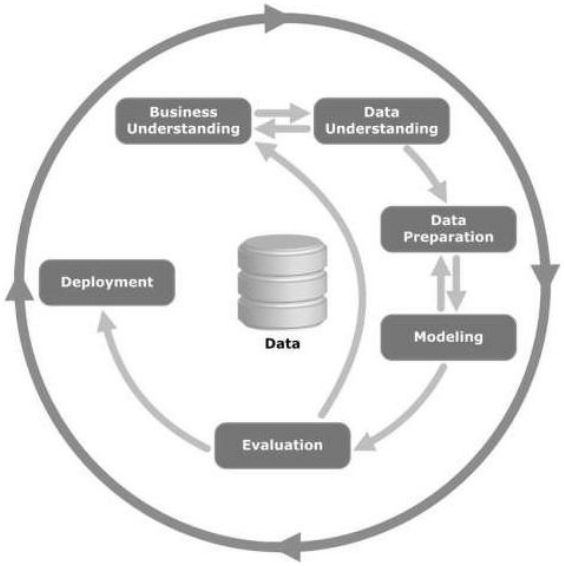


Figure 1.3 major stages of the cycle as described by the CRISP-DM methodology

Data Understanding − The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

Data Preparation − The data preparation phase covers all activities to construct the final dataset (data that will be fed into the modeling tool(s)) from the initial raw data. Data preparation tasks are likely to be performed multiple times, and not in any prescribed

order. Tasks include table, record, and attribute selection as well as transformation and cleaning of data for modeling tools.

Modeling − In this phase, various modeling techniques are selected and applied and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type. Some techniques have specific requirements on the form of data. Therefore, it is often required to step back to the data preparation phase.

Evaluation − At this stage in the project, you have built a model (or models) that appears to have high quality, from a data analysis perspective. Before proceeding to final deployment of the model, it is important to evaluate the model thoroughly and review the steps executed to construct the model, to be certain it properly achieves the business objectives.
A key objective is to determine if there is some important business issue that has not been sufficiently considered. At the end of this phase, a decision on the use of the data mining results should be reached.

Deployment − Creation of the model is generally not the end of the project. Even if the purpose of the model is to increase knowledge of the data, the knowledge gained will need to be organized and presented in a way that is useful to the customer.
Depending on the requirements, the deployment phase can be as simple as generating a report or as complex as implementing a repeatable data scoring (e.g. segment allocation) or data mining process.

In many cases, it will be the customer, not the data analyst, who will carry out the deployment steps. Even if the analyst deploys the model, it is important for the customer to understand upfront the actions which will need to be carried out in order to actually make use of the created models.

SEMMA Methodology
SEMMA is another methodology developed by SAS for data mining modeling. It stands for Sample, Explore, Modify, Model, and Asses. Here is a brief description of its stages −

Sample − The process starts with data sampling, e.g., selecting the dataset for modeling. The dataset should be large enough to contain sufficient information to retrieve, yet small enough to be used efficiently. This phase also deals with data partitioning.

Explore − This phase covers the understanding of the data by discovering anticipated and unanticipated relationships between the variables, and also abnormalities, with the help of data visualization.

Modify − The Modify phase contains methods to select, create and transform variables in preparation for data modeling.

Model − In the Model phase, the focus is on applying various modeling (data mining) techniques on the prepared variables in order to create models that possibly provide the desired outcome.

Assess − The evaluation of the modeling results shows the reliability and usefulness of the created models.

The main difference between CRISM–DM and SEMMA is that SEMMA focuses on the modeling aspect, whereas CRISP-DM gives more importance to stages of the cycle prior to modeling such as understanding the business problem to be solved, understanding and preprocessing the data to be used as input, for example, machine learning algorithms.

1.3.2 Big Data Life Cycle [2]

In today's big data context, the previous approaches are either incomplete or suboptimal. For example, the SEMMA methodology disregards completely data collection and preprocessing of different data sources. These stages normally constitute most of the work in a successful big data project.

A big data analytics cycle can be described by the following stage

- Business Problem Definition
- Research
- Human Resources Assessment
- Data Acquisition
- Data Munging
- Data Storage
- Exploratory Data Analysis
- Data Preparation for Modeling and Assessment
- Modeling
- Implementation

Business Problem Definition

This is a point common in traditional BI and big data analytics life cycle. Normally it is a non-trivial stage of a big data project to define the problem and evaluate correctly how much potential gain it may have for an organization. It seems obvious to mention this, but it has to be evaluated what are the expected gains and costs of the project.

Research

Analyze what other companies have done in the same situation. This involves looking for solutions that are reasonable for your company, even though it involves adapting other solutions to the resources and requirements that your company has. In this stage, a methodology for the future stages should be defined.

Human Resources Assessment

Once the problem is defined, it's reasonable to continue analyzing if the current staff is able to complete the project successfully. Traditional BI teams might not be capable to deliver an optimal solution to all the stages, so it should be considered before starting the project if there is a need to outsource a part of the project or hire more people.

Data Acquisition

This section is key in a big data life cycle; it defines which type of profiles would be needed to deliver the resultant data product. Data gathering is a non-trivial step of the process; it normally involves gathering unstructured data from different sources. To give an example, it could involve writing a crawler to retrieve reviews from a website. This involves dealing with text, perhaps in different languages normally requiring a significant amount of time to be completed.

Data Munging

Once the data is retrieved, for example, from the web, it needs to be stored in an easyto-use format. To continue with the reviews examples, let's assume the data is retrieved from different sites where each has a different display of the data.

Suppose one data source gives reviews in terms of rating in stars, therefore it is possible to read this as a mapping for the response variable $y \in \{1, 2, 3, 4, 5\}$. Another data source gives reviews using two arrows system, one for up voting and the other for down voting. This would imply a response variable of the form $y \in \{\text{positive, negative}\}$.

In order to combine both the data sources, a decision has to be made in order to make these two response representations equivalent. This can involve converting the first data source response representation to the second form, considering one star as negative and five stars as positive. This process often requires a large time allocation to be delivered with good quality.

Data Storage

Once the data is processed, it sometimes needs to be stored in a database. Big data technologies offer plenty of alternatives regarding this point. The most common alternative is using the Hadoop File System for storage that provides users a limited version of SQL, known as HIVE Query Language. This allows most analytics task to be

done in similar ways as would be done in traditional BI data warehouses, from the user perspective. Other storage options to be considered are MongoDB, Redis, and SPARK.
This stage of the cycle is related to the human resources knowledge in terms of their abilities to implement different architectures. Modified versions of traditional data warehouses are still being used in large scale applications. For example, teradata and IBM offer SQL databases that can handle terabytes of data; open source solutions such as postgreSQL and MySQL are still being used for large scale applications.

Even though there are differences in how the different storages work in the background, from the client side, most solutions provide a SQL API. Hence having a good understanding of SQL is still a key skill to have for big data analytics.

This stage a priori seems to be the most important topic, in practice, this is not true. It is not even an essential stage. It is possible to implement a big data solution that would be working with real-time data, so in this case, we only need to gather data to develop the model and then implement it in real time. So there would not be a need to formally store the data at all.

Exploratory Data Analysis
Once the data has been cleaned and stored in a way that insights can be retrieved from it, the data exploration phase is mandatory. The objective of this stage is to understand the data, this is normally done with statistical techniques and also plotting the data. This is a good stage to evaluate whether the problem definition makes sense or is feasible.

Data Preparation for Modeling and Assessment
This stage involves reshaping the cleaned data retrieved previously and using statistical preprocessing for missing values imputation, outlier detection, normalization, feature extraction and feature selection.

Modelling
The prior stage should have produced several datasets for training and testing, for example, a predictive model. This stage involves trying different models and looking forward to solving the business problem at hand. In practice, it is normally desired that the model would give some insight into the business. Finally, the best model or combination of models is selected evaluating its performance on a left-out dataset.

Implementation
In this stage, the data product developed is implemented in the data pipeline of the company. This involves setting up a validation scheme while the data product is working, in order to track its performance. For example, in the case of implementing a predictive

model, this stage would involve applying the model to new data and once the response is available, evaluate the model.

1.4. BIG DATA ANALYTICS - METHODOLOGY [2]

In terms of methodology, big data analytics differs significantly from the traditional statistical approach of experimental design. Analytics starts with data. Normally we model the data in a way to explain a response. The objective of this approach is to predict the response behavior or understand how the input variables relate to a response. Normally in statistical experimental designs, an experiment is developed and data is retrieved as a result. This allows to generate data in a way that can be used by a statistical model, where certain assumptions hold such as independence, normality, and randomization.

In big data analytics, we are presented with the data. We cannot design an experiment that fulfills our favorite statistical model. In large-scale applications of analytics, a large amount of work (normally 80% of the effort) is needed just for cleaning the data, so it can be used by a machine learning model.

We don't have a unique methodology to follow in real large-scale applications. Normally once the business problem is defined, a research stage is needed to design the methodology to be used. However general guidelines are relevant to be mentioned and apply to almost all problems.

One of the most important tasks in big data analytics is statistical modeling, meaning supervised and unsupervised classification or regression problems. Once the data is cleaned and preprocessed, available for modeling, care should be taken in evaluating different models with reasonable loss metrics and then once the model is implemented, further evaluation and results should be reported. A common pitfall in predictive modeling is to just implement the model and never measure its performance.

1.5. ANALYTICS PROCESS MODEL [3]

Figure 4 gives a high-level overview of the analytics process model.[4] As a first step, a thorough definition of the business problem to be solved with analytics is needed. Next, all source data need to be identified that could be of potential interest. This is a very important step, as data is the key ingredient to any analytical exercise and the selection of data will have a deterministic impact on the analytical models that will be built in a subsequent step. All data will then be gathered in a staging area, which could be, for

example, a data mart or data warehouse. Some basic exploratory analysis can be considered here using, for example, online analytical processing (OLAP) facilities for multidimensional data analysis (e.g., roll-up, drill down, slicing and dicing). This will be followed by a data cleaning step to get rid of all inconsistencies, such as missing values, outliers, and duplicate data. Additional transformations may also be considered, such as binning, alphanumeric to numeric coding, geographical aggregation, and so forth. In the analytics step, an analytical model will be estimated on the preprocessed and transformed data. Different types of analytics can be considered here (e.g., to do churn prediction, fraud detection, customer segmentation, market basket analysis). Finally, once the model has been built, it will be interpreted and evaluated by the business experts. Usually, many trivial patterns will be detected by the model. For example, in a market basket analysis setting, one may find that spaghetti and spaghetti sauce are often purchased together. These patterns are interesting because they provide some validation of the model. But of course, the key issue here is to find the unexpected yet interesting and actionable patterns (sometimes also referred to as knowledge diamonds) that can provide added value in the business setting. Once the analytical model has been appropriately validated and approved, it can be put into production as an analytics application (e.g., decision support system, scoring engine). It is important to consider here how to represent the model output in a user-friendly way, how to integrate it with other applications (e.g., campaign management tools, risk engines), and how to make sure the analytical model can be appropriately monitored and back tested on an ongoing basis.

It is important to note that the process model outlined in Figure 1.4 is iterative in nature, in the sense that one may have to go back to previous steps during the exercise. For example, during the analytics step, the need for additional data may be identified, which may necessitate additional cleaning, transformation, and so forth. Also, the most time consuming step is the data selection and preprocessing step; this usually takes around 80% of the total efforts needed to build an analytical model.

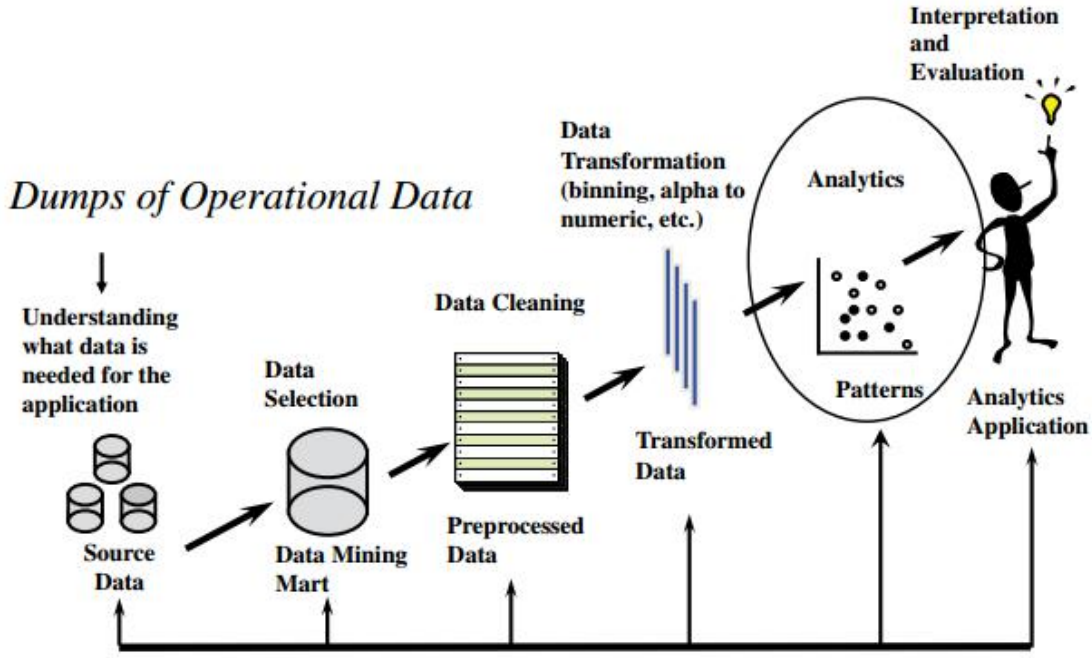


Figure 1.4 The Analytics Process Model [3]

1.6. ANALYTICAL MODEL REQUIREMENTS [3]

A good analytical model should satisfy several requirements, depending on the application area. A first critical success factor is business relevance. The analytical model should actually solve the business problem for which it was developed. It makes no sense to have a working analytical model that got sidetracked from the original problem statement. In order to achieve business relevance, it is of key importance that the business problem to be solved is appropriately defined, qualified, and agreed upon by all parties involved at the outset of the analysis.

A second criterion is statistical performance. The model should have statistical significance and predictive power. How this can be measured will depend upon the type of analytics considered. For example, in a classification setting (churn, fraud), the model should have good discrimination power. In a clustering setting, the clusters should be as homogenous as possible. In later chapters, we will extensively discuss various measures to quantify this.

Depending on the application, analytical models should also be interpretable and justifiable. Interpretability refers to understanding the patterns that the analytical model captures. This aspect has a certain degree of subjectivism, since interpretability may depend on the business user's knowledge. In many settings, however, it is considered to be a key requirement. For example, in credit risk modeling or medical diagnosis, interpretable models are absolutely needed to get good insight into the underlying data

patterns. In other settings, such as response modeling and fraud detection, having interpretable models may be less of an issue. Justifiability refers to the degree to which a model corresponds to prior business knowledge and intuition.[5] For example, a model stating that a higher debt ratio results in more creditworthy clients may be interpretable, but is not justifiable because it contradicts basic financial intuition. Note that both interpretability and justifiability often need to be balanced against statistical performance. Often one will observe that high performing analytical models are incomprehensible and black box in nature.

A popular example of this is neural networks, which are universal approximators and are high performing, but offer no insight into the underlying patterns in the data. On the contrary, linear regression models are very transparent and comprehensible, but offer only limited modeling power.

Analytical models should also be operationally efficient. This refers to the efforts needed to collect the data, preprocess it, evaluate the model, and feed its outputs to the business application (e.g., campaign management, capital calculation). Especially in a real-time online scoring environment (e.g., fraud detection) this may be a crucial characteristic. Operational efficiency also entails the efforts needed to monitor and backtest the model, and reestimate it when necessary. Another key attention point is the economic cost needed to set up the analytical model. This includes the costs to gather and preprocess the data, the costs to analyze the data, and the costs to put the resulting analytical models into production. In addition, the software costs and human and computing resources should be taken into account here. It is important to do a thorough cost–benefit analysis at the start of the project.

Finally, analytical models should also comply with both local and international regulation and legislation. For example, in a credit risk setting, the Basel II and Basel III Capital Accords have been introduced to appropriately identify the types of data that can or cannot be used to build credit risk models. In an insurance setting, the Solvency II Accord plays a similar role. Given the importance of analytics nowadays, more and more regulation is being introduced relating to the development and use of the analytical models. In addition, in the context of privacy, many new regulatory developments are taking place at various levels. A popular example here concerns the use of cookies in a web analytics context.

1.7. DATA COLLECTION, SAMPLING, AND PREPROCESSING [3]

Data are key ingredients for any analytical exercise. Hence, it is important to thoroughly consider and list all data sources that are of potential interest before starting the analysis. The rule here is the more data, the better. However, real life data can be dirty because of inconsistencies, incompleteness, duplication, and merging problems.

Throughout the analytical modeling steps, various data filtering mechanisms will be applied to clean up and reduce the data to a manageable and relevant size. Worth mentioning here is the garbage in, garbage out (GIGO) principle, which essentially states that messy data will yield messy analytical models. It is of the utmost importance that every data preprocessing step is carefully justified, carried out, validated, and documented before proceeding with further analysis. Even the slightest mistake can make the data totally unusable for further analysis. In what follows, we will elaborate on the most important data preprocessing steps that should be considered during an analytical modeling exercise.

1.7.1 Types of Data Sources [3]

As previously mentioned, more data is better to start off the analysis. Data can originate from a variety of different sources, which will be explored in what follows.

Transactions are the first important source of data. Transactional data consist of structured, low-level, detailed information capturing the key characteristics of a customer transaction (e.g., purchase, claim, cash transfer, credit card payment). This type of data is usually storedin massive online transaction processing (OLTP) relational databases.

It can also be summarized over longer time horizons by aggregating it into averages, absolute/relative trends, maximum/minimum values, and so on.

Unstructured data embedded in text documents (e.g., emails, web pages, claim forms) or multimedia content can also be interesting to analyze. However, these sources typically require extensive preprocessing before they can be successfully included in an analytical exercise.

Another important source of data is qualitative, expert-based data. An expert is a person with a substantial amount of subject matter expertise within a particular setting (e.g., credit portfolio manager, brand manager). The expertise stems from both common sense and business experience, and it is important to elicit expertise as much as possible before the analytics is run. This will steer the modeling in the right direction and allow you to interpret the analytical results from the right perspective. A popular example of applying expert-based validation is checking the univariate signs of a regression model. For example, one would expect a priori that higher debt has an adverse impact on credit risk, such that it should have a negative sign in the final scorecard. If this turns out not to be the case (e.g., due to bad data quality, multicollinearity), the expert/business user will not be tempted to use the analytical model at all, since it contradicts prior expectations.

Nowadays, data poolers are becoming more and more important in the industry. Popular examples are Dun & Bradstreet, Bureau Van Dijck, and Thomson Reuters. The core business of these companies is to gather data in a particular setting (e.g., credit risk, marketing), build models with it, and sell the output of these models (e.g., scores), possibly together with the underlying raw data, to interested customers. A popular

example of this in the United States is the FICO score, which is a credit score ranging between 300 and 850 that is provided by the three most important credit bureaus: Experian, Equifax, and Transunion. Many financial institutions use these FICO scores either as their final internal model or as a benchmark against an internally developed credit scorecard to better understand the weaknesses of the latter.

Finally, plenty of publicly available data can be included in the analytical exercise. A first important example is macroeconomic data about gross domestic product (GDP), inflation, unemployment, and so on. By including this type of data in an analytical model, it will become possible to see how the model varies with the state of the economy.

This is especially relevant in a credit risk setting, where typically all models need to be thoroughly stress tested. In addition, social media data from Facebook, Twitter, and others can be an important source of information. However, one needs to be careful here and make sure that all data gathering respects both local and international privacy regulations.

1.7.2 Sampling [3]

The aim of sampling is to take a subset of past customer data and use that to build an analytical model. A first obvious question concerns the need for sampling. With the availability of high performance computing facilities (e.g., grid/cloud computing), one could also directly analyze the full data set. However, a key requirement for a good sample is that it should be representative of the future customers on which the analytical model will be run. Hence, the timing aspect becomes important because customers of today are more similar to customers of tomorrow than customers of yesterday. Choosing the optimal time window for the sample involves a trade-off between lots of data (and hence a more robust analytical model) and recent data (which may be more representative). The sample should also be taken from an average business period to get a picture of the target population that is as accurate as possible.

It speaks for itself that sampling bias should be avoided as much as possible. However, this is not always straightforward. Let's take the example of credit scoring. Assume one wants to build an application scorecard to score mortgage applications. The future population then consists of all customers who come to the bank and apply for a mortgage—the so-called through-the-door (TTD) population. One then needs a subset of the historical TTD population to build an analytical model. However, in the past, the bank was already applying a credit policy (either expert based or based on a previous analytical model). This implies that the historical TTD population has two subsets: the customers that were accepted with the old policy and the ones that were rejected (see Figure 1.5). Obviously, for the latter, we don't know the target value since they were never granted the credit. When building a sample, one can then only make use of those that were accepted, which clearly implies a bias. Procedures for reject inference have

been suggested in the literature to deal with this sampling bias problem.[6] Unfortunately, all of these procedures make assumptions and none of them works perfectly. One of the most popular solutions is bureau-based inference, whereby a sample of past customers is given to the credit bureau to determine their target label (good or bad payer).

When thinking even closer about the target population for credit scoring, another forgotten subset is the withdrawals. These are the customers who were offered credit but decided not to take it (despite the fact that they may have been classified as good by the old scorecard). To be representative, these customers should also be included in the development sample. However, to the best of our knowledge, no procedures for withdrawal inference are typically applied in the industry.

In stratified sampling, a sample is taken according to predefined strata. Consider, for example, a churn prediction or fraud detection context in which data sets are typically very skewed (e.g., 99 percent nonchurners and 1 percent churners). When stratifying according to the target churn indicator, the sample will contain exactly the same percentages of churners and nonchurners as in the original data.

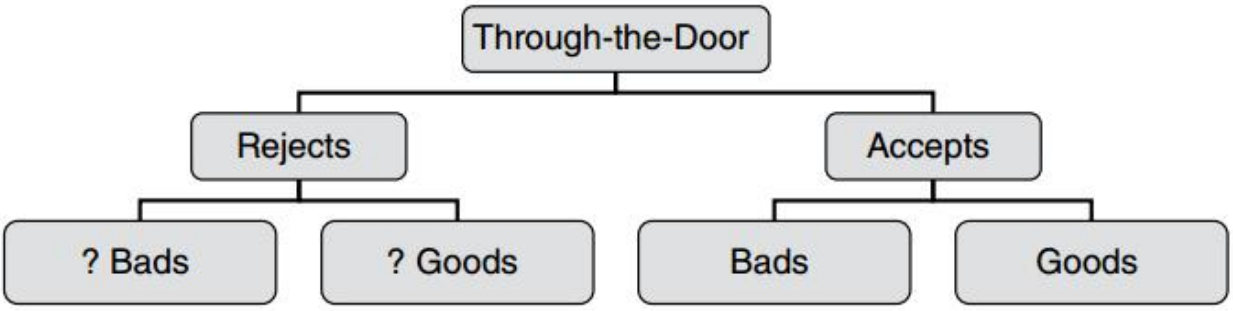


Figure 1.5 The Reject Inference Problem in Credit Scoring [3]

1.7.3 Types of data elements [3]

It is important to appropriately consider the different types of data elements at the start of the analysis. The following types of data elements can be considered:

- *Continuous*: These are data elements that are defined on an interval that can be limited or unlimited. Examples include income, sales, RFM (recency, frequency, monetary).
- *Categorical*
  - *Nominal*: These are data elements that can only take on a limited set of values with no meaningful ordering in between.
    Examples include marital status, profession, purpose of loan.
  - *Ordinal*: These are data elements that can only take on a limited set of values with a meaningful ordering in between. Examples include credit rating; age coded as young, middle aged, and old.

- *Binary*: These are data elements that can only take on two values. Examples include gender, employment status.

Appropriately distinguishing between these different data elements is of key importance to start the analysis when importing the data into an analytics tool. For example, if marital status were to be incorrectly specified as a continuous data element, then the software would calculate its mean, standard deviation, and so on, which is obviously meaningless.

1.7.4 Visual data exploration and exploratory statistical analysis [3]

Visual data exploration is a very important part of getting to know your data in an "informal" way. It allows you to get some initial insights into the data, which can then be usefully adopted throughout the modeling. Different plots/graphs can be useful here. A first popular example is pie charts. A pie chart represents a variable's distribution as a pie, whereby each section represents the portion of the total percent taken by each value of the variable. Figure 1.6 represents a pie chart for a housing variable for which one's status can be own, rent, or for free (e.g., live with parents). By doing a separate pie chart analysis for the goods and bads, respectively, one can see that more goods own their residential property than bads, which can be a very useful starting insight. Bar charts represent the frequency of each of the values (either absolute or relative) as bars. Other handy visual tools are histograms and scatter plots. A histogram provides an easy way to visualize the central tendency and to determine the variability or spread of the data. It also allows you to contrast the observed data with standard known distributions (e.g., normal distribution). Scatter plots allow you to visualize one variable against another to see whether there are any correlation patterns in the data. Also, OLAP-based multidimensional data analysis can be usefully adopted to explore patterns in the data.

A next step after visual analysis could be inspecting some basic statistical measurements, such as averages, standard deviations, minimum, maximum, percentiles, and confidence intervals. One could calculate these measures separately for each of the target classes (e.g., good versus bad customer) to see whether there are any interesting patterns present (e.g., whether bad payers usually have a lower average age than good payers).

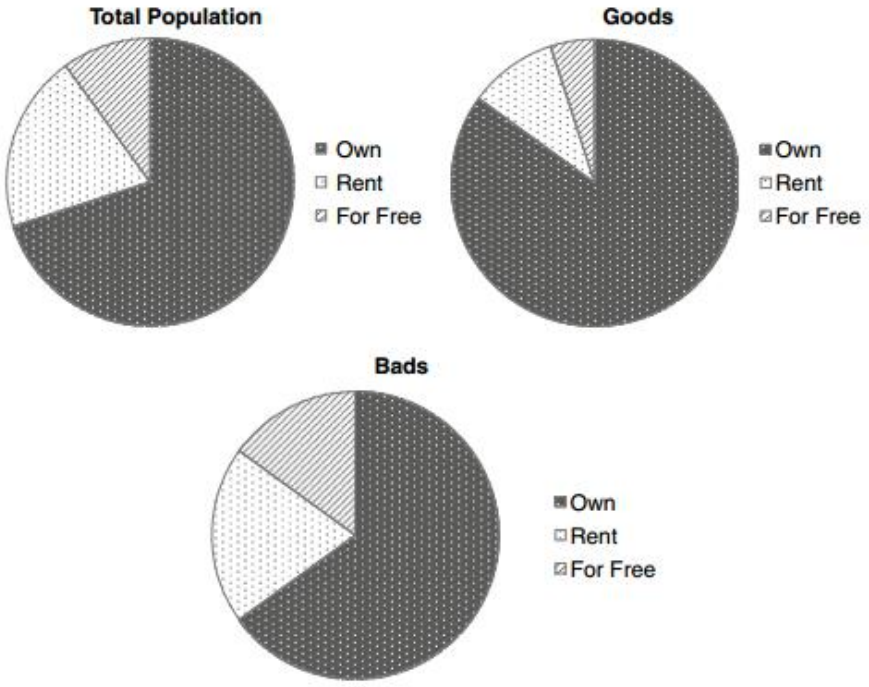


Figure 1.6 Pie Charts for Exploratory Data Analysis [3]

1.7.5 Missing values [3]

Missing values can occur because of various reasons. The information can be nonapplicable. For example, when modeling time of churn, this information is only available for the churners and not for the nonchurners because it is not applicable there. The information can also be undisclosed. For example, a customer decided not to disclose his or her income because of privacy. Missing data can also originate because of an error during merging (e.g., typos in name or ID).

Some analytical techniques (e.g., decision trees) can directly deal with missing values. Other techniques need some additional preprocessing. The following are the most popular schemes to deal with missing values:[7]

- *Replace (impute)*. This implies replacing the missing value with a known value (e.g., consider the example in Table 1.1). One could impute the missing credit bureau scores with the average or median of the known values. For marital status, the mode can then be used. One could also apply regression-based imputation whereby a regression model is estimated to model a target variable (e.g., credit bureau score) based on the other information available (e.g., age, income). The latter is more sophisticated, although the added value from an empirical viewpoint (e.g., in terms of model performance) is questionable.
- *Delete*. This is the most straightforward option and consists of deleting observations or variables with lots of missing values. This, of course, assumes

that information is missing at random and has no meaningful interpretation and/or relationship to the target.

- *Keep*. Missing values can be meaningful (e.g., a customer did not disclose his or her income because he or she is currently unemployed). Obviously, this is clearly related to the target (e.g., good/bad risk or churn) and needs to be considered as a separate category.

As a practical way of working, one can first start with statistically testing whether missing information is related to the target variable (using, for example, a chi-squared test, discussed later). If yes, then we can adopt the keep strategy and make a special category for it. If not, one can, depending on the number of observations available, decide to either delete or impute.

ID	Age	Income	Marital Status	Credit Bureau Score	Class
1	34	1,800	?	620	Churner
2	28	1,200	Single	?	Nonchurner
3	22	1,000	Single	?	Nonchurner
4	60	2,200	Widowed	700	Churner
5	58	2,000	Married	?	Nonchurner
6	44	?	?	?	Nonchurner
7	22	1,200	Single	?	Nonchurner
8	26	1,500	Married	350	Nonchurner
9	34	?	Single	?	Churner
10	50	2,100	Divorced	?	Nonchurner

Table 1.1 Dealing with Missing Values [3]

1.7.6 Outlier detection and treatment [3]

Outliers are extreme observations that are very dissimilar to the rest of the population. Actually, two types of outliers can be considered:

1. Valid observations (e.g., salary of boss is $1 million)
2. Invalid observations (e.g., age is 300 years)

Both are univariate outliers in the sense that they are outlying on one dimension. However, outliers can be hidden in unidimensional views of the data. Multivariate outliers are observations that are outlying in multiple dimensions. Figure 1.7 gives an example of two outlying observations considering both the dimensions of income and age. Two important steps in dealing with outliers are detection and treatment. A first obvious check for outliers is to calculate the minimum and maximum values for each of the data

elements. Various graphical tools can be used to detect outliers. Histograms are a first example.

Figure 1.8 presents an example of a distribution for age whereby the circled areas clearly represent outliers.

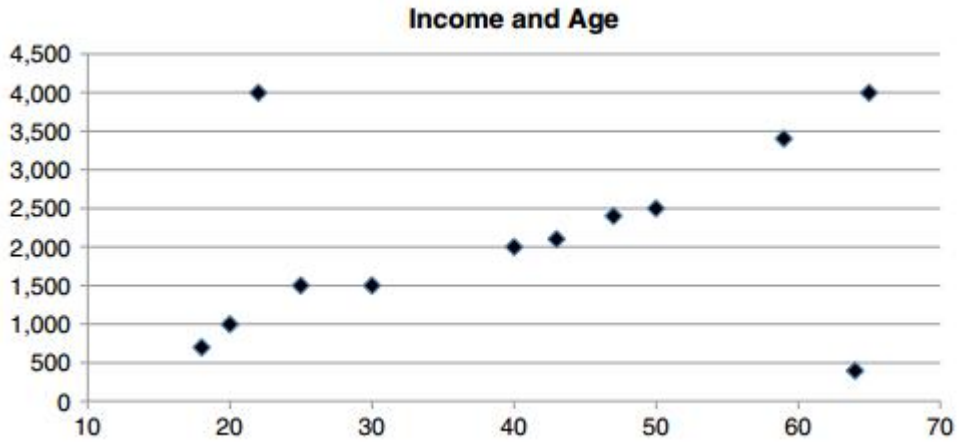


Figure 1.7 Multivariate Outliers [3]

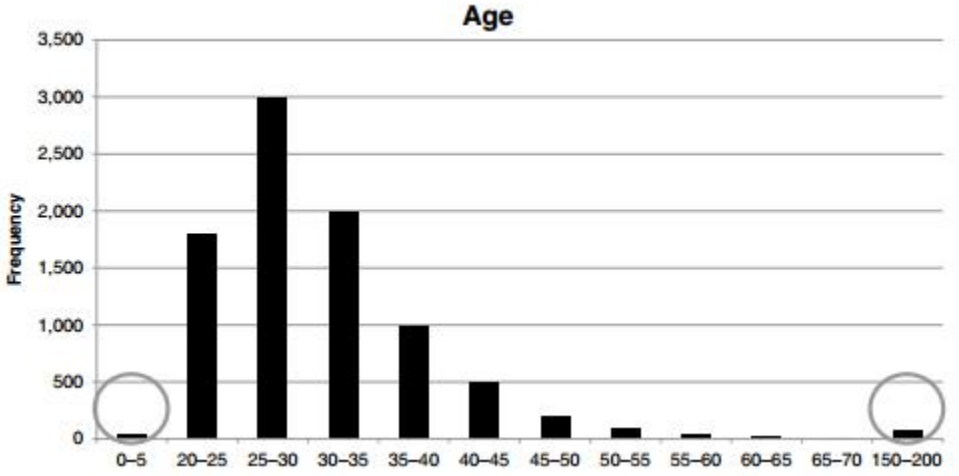


Figure 1.8 Histograms for Outlier Detection [3]

Another useful visual mechanism are box plots. A box plot represents three key quartiles of the data: the first quartile (25 percent of the observations have a lower value), the median (50 percent of the observations have a lower value), and the third quartile (75 percent of the observations have a lower value). All three quartiles are represented as a box. The minimum and maximum values are then also added unless they are too far away from the edges of the box. Too far away is then quantified as more than 1.5 * Interquartile Range (IQR = Q3 − Q1). Figure 1.9 gives an example of a box plot in which three outliers can be seen.

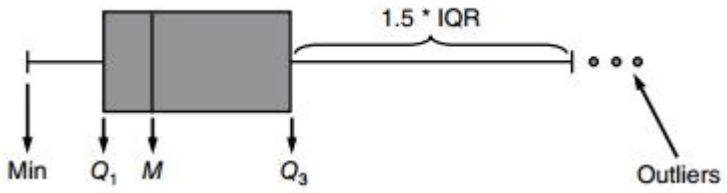


Figure 1.9 Box Plots for Outlier Detection [3]

Another way is to calculate z-scores, measuring how many standard deviations an observation lies away from the mean, as follows:

$$z_i = \frac{x_i - \mu}{\sigma}$$

where μ represents the average of the variable and σ its standard deviation. An example is given in Table 1.2. Note that by definition, the z-scores will have 0 mean and unit standard deviation.

A practical rule of thumb then defines outliers when the absolute value of the z-score |z| is bigger than 3. Note that the z-score relies on the normal distribution.

ID	Age	Z-Score
1	30	(30 – 40)/10 = –1
2	50	(50 – 40)/10 = +1
3	10	(10 – 40)/10 = –3
4	40	(40 – 40)/10 = 0
5	60	(60 – 40)/10 = +2
6	80	(80 – 40)/10 = +4
…	…	…
	μ = 40 σ = 10	μ = 0 σ = 1

Table 1.2 Z-Scores for Outlier Detection [3]

The above methods all focus on univariate outliers. Multivariate outliers can be detected by fitting regression lines and inspecting the observations with large errors (using, for example, a residual plot).

Alternative methods are clustering or calculating the Mahalanobis distance. Note, however, that although potentially useful, multivariate outlier detection is typically not considered in many modeling exercises due to the typical marginal impact on model performance.

Some analytical techniques (e.g., decision trees, neural networks, Support Vector Machines (SVMs)) are fairly robust with respect to outliers. Others (e.g., linear/logistic regression) are more sensitive to them. Various schemes exist to deal with outliers. It highly depends on whether the outlier represents a valid or invalid observation. For invalid observations (e.g., age is 300 years), one could treat the outlier as a missing value using any of the schemes discussed in the previous section. For valid observations (e.g., income is $1 million), other schemes are needed. A popular scheme is truncation/capping/winsorizing. One hereby imposes both a lower and upper limit on a variable and any values below/above are brought back to these limits. The limits can be calculated using the z-scores (see Figure 1.10), or the IQR (which is more robust than the z-scores), as follows:

$$\text{Upper/lower limit} = M \pm 3s, \text{ with } M = \text{median and } s = IQR/(2 \times 0.6745).^{3}$$

A sigmoid transformation ranging between 0 and 1 can also be used for capping, as follows:

$$f(x) = \frac{1}{1+e^{-x}}$$

In addition, expert-based limits based on business knowledge and/or experience can be imposed.

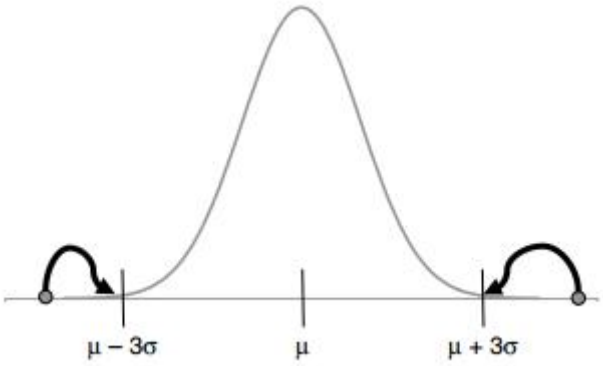


Figure 1.10 Using the Z-Scores for Truncation [3]

1.7.7 Categorization [3]

Categorization (also known as coarse classification, classing, grouping, binning, etc.) can be done for various reasons. For categorical variables, it is needed to reduce the number of categories. Consider, for example, the variable "purpose of loan" having 50 different values.

When this variable would be put into a regression model, one would need 49 dummy variables (50 − 1 because of the collinearity), which would necessitate the estimation of

49 parameters for only one variable. With categorization, one would create categories of values such that fewer parameters will have to be estimated and a more robust model is obtained.

For continuous variables, categorization may also be very beneficial. Consider, for example, the age variable and its risk as depicted in Figure 1.11. Clearly, there is a nonmonotonous relation between risk and age. If a nonlinear model (e.g., neural network, support vector machine) were to be used, then the nonlinearity can be perfectly modeled. However, if a regression model were to be used (which is typically more common because of its interpretability), then since it can only fit a line, it will miss out on the nonmonotonicity. By categorizing the variable into ranges, part of the nonmonotonicity can be taken into account in the regression. Hence, categorization of continuous variables can be useful to model nonlinear effects into linear models.

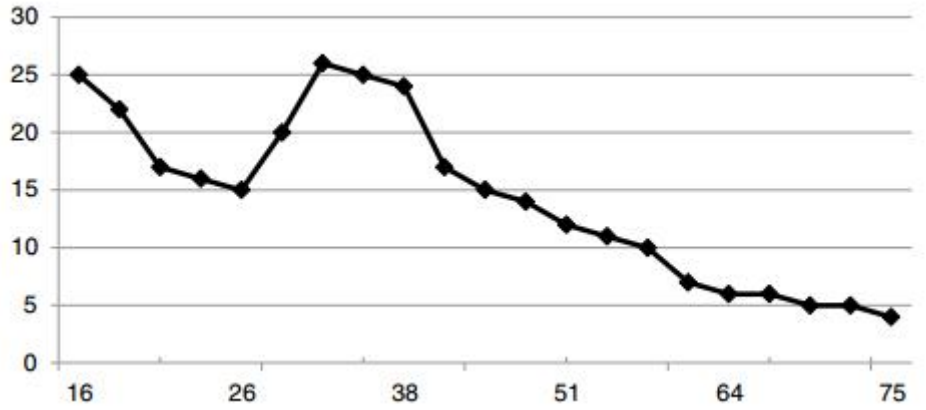


Figure 1.11 Default Risk versus Age [3]

Various methods can be used to do categorization. Two very basic methods are equal interval binning and equal frequency binning.

Consider, for example, the income values 1,000, 1,200, 1,300, 2,000, 1,800, and 1,400. Equal interval binning would create two bins with the same range—Bin 1: 1,000, 1,500 and Bin 2: 1,500, 2,000—whereas equal frequency binning would create two bins with the same number of observations—Bin 1: 1,000, 1,200, 1,300; Bin 2: 1,400, 1,800, 2,000. However, both methods are quite basic and do not take into account a target variable (e.g., churn, fraud, credit risk).

Chi-squared analysis is a more sophisticated way to do coarse classification. Consider the example depicted in Table 1.3 for coarse classifying a residential status variable.

Attribute	Owner	Rent Unfurnished	Rent Furnished	With Parents	Other	No Answer	Total
Goods	6,000	1,600	350	950	90	10	9,000
Bads	300	400	140	100	50	10	1,000
Good: bad odds	20:1	4:1	2.5:1	9.5:1	1.8:1	1:1	9:1

Source: L. C. Thomas, D. Edelman, and J. N. Crook, *Credit Scoring and its Applications* (Society for Industrial and Applied Mathematics, Philadelphia, Penn., 2002).

Table 1.3 Coarse Classifying the Residential Status Variable

Suppose we want three categories and consider the following options:

- Option 1: owner, renters, others
- Option 2: owner, with parents, others

Both options can now be investigated using chi-squared analysis.

The purpose is to compare the empirically observed with the independence frequencies. For option 1, the empirically observed frequencies are depicted in Table 1.4.

The independence frequencies can be calculated as follows. The number of good owners, given that the odds are the same as in the whole population, is 6,300/10,000 × 9,000/10,000 × 10,000 = 5,670. One then obtains Table 1.5.

The more the numbers in both tables differ, the less independence, hence better dependence and a better coarse classification. Formally, one can calculate the chi-squared distance as follows:

$$\chi^2 = \frac{(6000-5670)^2}{5670} + \frac{(300-630)^2}{630} + \frac{(1950-2241)^2}{2241} + \frac{(540-249)^2}{249} + \frac{(1050-1089)^2}{1089} + \frac{(160-121)^2}{121} = 583$$

Attribute	Owner	Renters	Others	Total
Goods	6,000	1,950	1,050	9,000
Bads	300	540	160	1,000
Total	6,300	2,490	1,210	10,000

Table 1.4 Empirical Frequencies Option 1 for Coarse Classifying Residential Status [3]

Attribute	Owner	Renters	Others	Total
Goods	5,670	2,241	1,089	9,000
Bads	630	249	121	1,000
Total	6,300	2,490	1,210	10,000

Table 1.5 Independence Frequencies Option 1 for Coarse Classifying Residential Status [3]

Likewise, for option 2, the calculation becomes:

$$\chi^2 = \frac{(6000-5670)^2}{5670} + \frac{(300-630)^2}{630} + \frac{(950-945)^2}{945} + \frac{(100-105)^2}{105} + \frac{(2050-2385)^2}{2385} + \frac{(600-265)^2}{265} = 662$$

So, based upon the chi-squared values, option 2 is the better categorization. Note that formally, one needs to compare the value with a chi-squared distribution with k − 1 degrees of freedom with k being the number of values of the characteristic.

Many analytics software tools have built-in facilities to do categorization using chi-squared analysis. A very handy and simple approach (available in Microsoft Excel) is pivot tables. Consider the example shown in Table 1.6. One can then construct a pivot table and calculate the odds as shown in Table 1.7.

Customer ID	Age	Purpose	…	G/B
C1	44	Car		G
C2	20	Cash		G
C3	58	Travel		B
C4	26	Car		G
C5	30	Study		B
C6	32	House		G
C7	48	Cash		B
C8	60	Car		G
…	…	…		

Table 1.6 Coarse Classifying the Purpose Variable [3]

	Car	Cash	Travel	Study	House	…
Good	1,000	2,000	3,000	100	5,000	
Bad	500	100	200	80	800	
Odds	2	20	15	1.25	6.25	

Table 1.7 Pivot Table for Coarse Classifying the Purpose Variable [3]

We can then categorize the values based on similar odds. For example, category 1 (car, study), category 2 (house), and category 3(cash, travel).

1.7.8 Variable selection [3]

Many analytical modeling exercises start with tons of variables, of which typically only a few actually contribute to the prediction of the target variable. For example, the average application/behavioral scorecard in credit scoring has somewhere between 10 and 15 variables. The key question is how to find these variables. Filters are a very handy variable selection mechanism. They work by measuring univariate correlations between each variable and the target. As such, they allow for a quick screening of which variables should be retained for further analysis. Various filter measures have been suggested in the literature. One can categorize them as depicted in Table 1.8.
The Pearson correlation ρ_P is calculated as follows:

$$\rho_P = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

It measures a linear dependency between two variables and always varies between −1 and +1. To apply it as a filter, one could select all variables for which the Pearson correlation is significantly different from 0 (according to the p-value), or, for example, the ones where $|\rho_P| > 0.50$
The Fisher score can be calculated as follows:

$$\frac{\left|\bar{X}_G - \bar{X}_B\right|}{\sqrt{s_G^2 + s_B^2}},$$

where $\overline{X_G}$ ($\overline{X_B}$) represents the average value of the variable for the Goods (Bads) and s_G^2 (s_B^2) the corresponding variances. High values of the Fisher score indicate a predictive variable. To apply it as a filter, one could, for example, keep the top 10 percent. Note that the Fisher score may generalize to a well-known analysis of variance (ANOVA) in case a variable has multiple categories.
The information value (IV) filter is based on weights of evidence and is calculated as follows:

$$IV = \sum_{i=1}^{k}(Dist\ Good_i - Dist\ Bad_i) * WOE_i$$

where *k* represents the number of categories of the variable.

The following rules of thumb apply for the information value:

- < 0.02: unpredictive
- 0.02–0.1: weak predictive
- 0.1–0.3: medium predictive
- 0.3: strong predictive

Note that the information value assumes that the variable has been categorized. It can actually also be used to adjust/steer the categorization so as to optimize the IV. Many software tools will provide interactive support to do this, whereby the modeler can adjust the categories and gauge the impact on the IV. To apply it as a filter, one can calculate the information value of all (categorical) variables and only keep those for which the IV > 0.1 or, for example, the top 10%.

Another filter measure based upon chi-squared analysis is Cramer's V. Consider the contingency table depicted in Table 1.8 for marital status versus good/bad.

Similar to the example discussed in the section on categorization, the chi-squared value for independence can then be calculated as follows:

$$\chi^2 = \frac{(500-480)^2}{480} + \frac{(100-120)^2}{120} + \frac{(300-320)^2}{320} + \frac{(100-80)^2}{80} = 10.41$$

This follows a chi-squared distribution with k-1 degrees of freedom, with k being the number of classes of the characteristic. The Cramer's V measure can then be calculated as follows:

$$Cramer's\ V = \sqrt{\frac{\chi^2}{n}} = 0.10,$$

	Good	Bad	Total
Married	500	100	600
Not Married	300	100	400
Total	800	200	1,000

Table 1.8 Contingency Table for Marital Status versus Good/Bad Customer [3]

With n being the number of observations in the data set. Cramer's V is always bounded between 0 and 1 and higher values indicate better predictive power. As a rule of thumb, a cutoff of 0.1 is commonly adopted. One can then again select all variables where Cramer's V is bigger than 0.1, or consider the top 10 percent. Note that the information value and Cramer's V typically consider the same characteristics as most important.

Filters are very handy because they allow you to reduce the number of dimensions of the data set early in the analysis in a quick way.

Their main drawback is that they work univariately and typically do not consider, for example, correlation between the dimensions individually. Hence, a follow-up input selection step during the modeling phase will be necessary to further refine the characteristics. Also worth mentioning here is that other criteria may play a role in selecting variables. For example, from a regulatory compliance viewpoint, some variables may not be used in analytical models (e.g., the U.S. Equal Credit Opportunities Act states that one cannot discriminate credit based on age, gender, marital status, ethnic

origin, religion, and so on, so these variables should be left out of the analysis as soon as possible).
Note that different regulations may apply in different geographical regions and hence should be checked. Also, operational issues could be considered (e.g., trend variables could be very predictive but may require too much time to be computed in a real-time online scoring environment).

1.7.9 Segmentation [3]

Sometimes the data is segmented before the analytical modeling starts. A first reason for this could be strategic (e.g., banks might want to adopt special strategies to specific segments of customers).
It could also be motivated from an operational viewpoint (e.g., new customers must have separate models because the characteristics in the standard model do not make sense operationally for them).
Segmentation could also be needed to take into account significant variable interactions (e.g., if one variable strongly interacts with a number of others, it might be sensible to segment according to this variable).
The segmentation can be conducted using the experience and knowledge from a business expert, or it could be based on statistical analysis using, for example, decision trees, k-means, or self-organizing maps.
Segmentation is a very useful preprocessing activity because one can now estimate different analytical models each tailored to a specific segment. However, one needs to be careful with it because by segmenting, the number of analytical models to estimate will increase, which will obviously also increase the production, monitoring, and maintenance costs.

1.8. TYPES OF DATA [8]

Big Data, data science and business analytics work with structured and unstructured data. But SMART business occurs when we combine existing data sets with unstructured or semi-structured data from both internal and external sources.

Structured data
Structured data provides most of our current business insights but is often considered ‘old hat’ and a bit dull – especially in comparison to its rock star cousin, unstructured data – it is easy to ignore structured data. But that is a mistake as many Big Data insights are generated by combining structured and unstructured data.

Data that is located in a fixed field within a defined record or file is called structured data. This includes data contained in relational databases and spreadsheets.

Examples of structured data include:

- Point of sales data
- Financial data
- Customer data.

As the name would suggest structured data refers to data or information that has a predefined data model or is organized in a predetermined way.

A data model is a model of the types of business data that your business will record and how that data will be stored, processed and accessed. Within that data model the fields of data that you intend to capture need to be defined and any conventions set around how that data will be stored. For example, if you look at a standard customer database the fields that are defined will include name, address, contact telephone numbers, email address, etc. Within those fields conventions may also be set so, for example, the telephone number field will only accept numeric information. These conventions can also include drop down menus that limit the choices of the data that can be entered into a field, thus ensuring consistency of input. For example, a 'Title' field within a name structure may only give you certain options to choose from, such as Mr, Ms, Miss, Mrs, Dr, etc.
Structured data gives names to each field in a database and defines the relationships between the fields. As a result structured data is easy to input, easy to store and easy to analyze. Up until relatively recently technology just didn't have the grunt to store, never mind analyze, anything other than structured data.
Everything that didn't fit into the databases or spreadsheets was usually either discarded or stored on paper or microfiche in filing cabinets or storage facilities.
Structured data is often managed using Structured Query Language (SQL) – a programming language originally created by IBM in the 1970s for managing and querying data in relational database management systems. SQL represented a huge leap forward over paper-based data storage and analysis, but not everything in business will fit neatly into a predefined field.

Unstructured and semi-structured data

Unstructured and semi-structured data are like the popular kids at school! Everyone is talking about them and they represent the sexy new frontier lauded by Big Data. It is estimated that 80% of business-relevant information originates in unstructured or semistructured data.
It represents all the data that can't be so easily slotted into columns, rows and fields. It is usually text heavy, but may also contain data such as dates, numbers and facts or

different types of data such as images. These inconsistencies make it difficult to analyze using traditional computer programs.

Examples of unstructured and semi-structured data include:

- Photos and graphic images
- Videos
- Websites
- Text files or documents such as email, PDF, blogs, social media posts, etc.
- PowerPoint presentations.

Semi-structured data is a cross between unstructured and structured. This is data that may have some structure that can be used for analysis but lacks the strict data model structure. In semistructured data, tags or other types of markers are used to identify certain elements within the data, but the data doesn't have a rigid structure. For example, a Facebook post can be categorized by author, data, length and even sentiment but the content is generally unstructured. Another example is word processing software that includes metadata detailing the author's name, when it was created and amended but the content of the document is still unstructured.

Internal data

Internal data accounts for everything your business currently has or could access.
This includes private or proprietary data that is collected and owned by the business where you control access.

Examples of internal data include:

- Customer feedback
- Sales data
- Employee or customer survey data
- CCTV video data
- Transactional data
- Customer record data
- Stock control data
- HR data.

External data

External data is the infinite array of information that exists outside your business.
External data is either public or private. Public data is data that anyone can obtain – either by collecting it for free, paying a third party for it or getting a third party to collect it for you. Private data is usually something you would need to source and pay for from another business or third party data supplier.

Examples of external data include:

- Weather data
- Government data such as census data
- Twitter data
- Social media profile data
- Google Trends or Google Maps.

A lot of the Big Data hype focuses on unstructured data and the allure and promise of external data, often at the expense or dismissal of internal or structured data.
It is really important to understand that no type of data is inherently better or more valuable than any other type. The key is to start with your strategy and establish your SMART questions so that those questions guide you to the best structured, unstructured, internal or external data to answer those questions and deliver the strategy.
But before we explore how to do that let's take a moment to appreciate the new forms of data that are now at your disposal as you seek to answer those questions.

1.8.1 Datification: The new forms of data [8]

Most human and computer based activities already leave a digital trace (or data) that can be collected and analysed to provide insights on everything from health to crime to business performance. Of the few activities that don't currently leave a digital trace – they soon will.
The world is being 'datafied' and there are now many forms of useful data. Some of the data forms are new such as social media posts; others have been around for a long time. For example, we've been able to record conversations for a long time but a lack of storage capacity or a way to really analyse those recordings limited their utility. But all that is changing.

Data is now being mined from:

- Our activities
- Our conversations
- Photos and video
- Sensors
- The Internet of Things.

Activity data
More and more of the activities we engage in leave a data trail.

For example, when we go online our browser logs what we are searching for and what websites we visit. Most websites will log how many people visit the site, where those people are located (using the computer ISP), how long the person stayed on the site and how they navigated or clicked through the site. Often this information is used to assess website performance and delete areas that no-one visits while improving pages that seem to generate the most interest.

If we decide to go shopping online there is a record of what we share or like and of course what we buy, how much we paid for it, when we bought it, when it was delivered and often what we then thought of the product or service through user feedback.

If we decide to read a book chances are we will increasingly turn to a Kindle, iPad, smart phone or other e-reader. There are now millions of books available in a digital format. Some books such as technology text books which change rapidly are often never even released as a physical printed book.

It has been estimated that 130 million unique books have been published since the invention of the Gutenberg printing press in 1450.

By 2012, just seven years into the Google Book Project, Google had scanned over 20 million titles or more than 15% of the world's entire written heritage![9] Amazon also gives us extensive access to old books in digital form.

When we use an e-reader we are usually not just reading a digital image of the page – the text is datafied. That means that we can change font size, add notes, highlight text or search the book.

This datafication also means that data is gathered about what we read, how long we read for, whether we skip pages, what pages we annotate and what we choose to highlight. This information could certainly prove useful for authors and publishers. I would love to know how people use my books, which sections people skip, when readers stop reading a book. This would allow me – or indeed any author – to revise content in order to shorten or improve particular parts so that readers have a better experience. Furthermore, authors and publishers may be able to identify areas of interest from frequently highlighted passages across many books to identify new topic trends on which to commission new work.

If we listen to music using our smart phone or digital music player, data is also collected on what we are listening to, how long we are listening and what tracks we are skipping past. And artists like Lady Gaga are using this data to create playlists for live gigs and influence future song creation.

Even walking to work or going to the gym will generate data if we are wearing a smart device like the 'Up' band or are using an app on our smart phone. These apps and devices can measure how many steps we take each day, how many calories we burn, how well we sleep, log activity and exercise, deliver insights and celebrate milestones. Some devices also measure our heart rate and often our heart rate variation (HRV). HRV measures the tiny variations in the interval between each heart beat and has been proven to be a

significant metric for predicting health problems. For example, since 1965 it has been common obstetric practice to monitor a baby's HRV during labour for early signs of foetal distress.[10]

In 1997 Jacqueline Dekker, Professor of Diabetes Epidemiology at the VU University Medical Centre in Amsterdam, along with her colleagues discovered that HRV was capable of predicting death, not only in babies or heart attack victims but it also predicted 'all cause mortality'.[11] Clearly, data on HRV would be useful for us all to know and devices like smart watches will be able to collect such data.

Many of these wearable devices are now Internet-enabled so that they self-generate and share data. It is also almost inevitable that many of the current wearable devices and apps will be swallowed up by the smart watch in the same way iPods were swallowed up by iPhones.

The company that makes the 'Up' band, Jawbone, now collects sleep data from millions of people around the world. This means they have unparalleled access to years' worth of sleep data – every night! No company on the planet has ever had that sort of data or that sort of volume of data.

Jawbone is then able to analyse the data to understand more about sleep, our sleeping patterns and what disrupts those patterns. For example, Jawbone could look at the data and work out how many hours of sleep are lost, on average, when the Superbowl is broadcast in the US or how long it normally takes for travelers to get back to normal sleeping patterns if they fly between New York and San Francisco or between London and Sydney.

Conversation data

Increasingly we also leave digital records of our conversations – either through text when we write an SMS message, on social media or an audio recording of a telephone call.

Just think of the billions of emails that are sent and stored every week. In fact, twenty million emails were written in the time it took to read this sentence.[12]

We are using social media to communicate and interact with each other, which is creating unfathomable amounts of data. Check out these stats:

- More than a billion tweets are sent every 48 hours.
- One million accounts are added to Twitter every day.
- Every sixty seconds, 293,000 status updates are posted on Facebook.
- Two new members join LinkedIn every second (172,800 per day)
- 72% of online adults use social networking sites.
- 25 percent of Facebook users never bother with any kind of privacy control.
- The average Facebook user creates 90 pieces of content including links, news stories, photo albums, notes, and videos each month.
- Incredibly, people in New York City received tweets about the August 2011 earthquake in Mineral, Virginia 30 seconds before they felt it.[12]

There are also already millions of website and blogs contributing to the conversation. An estimated 571 new websites are created every minute of the day. Every minute, Tumblr owners publish approximately 27,778 new blog posts and 3 million new blogs come online every month.[12]

Plus there is the data collected from our telephone conversations. If you call a customer service department we are always told the conversation may be recorded. Often that data is being mined for content and sentiment and even analysed for stress levels in someone's voice to gauge how irritated the customers are!

Audio data is also being used to improve voice recognition and translation software. For example, Google decided to venture into translation in 2006 as part of its mission to 'organize the world's information and make it universally accessible and useful'. Most translation software utilize perfectly translated pages of text to create the algorithms but Google used the entire global Internet and more. Their system sucked in every translation – good and bad – that it could find in order to train the translation computers. As a result of the sheer volume of data that they could access and use Google translation is more accurate than any other system. By mid 2012 its dataset covered more than 60 languages and even accepts voice input in 14 languages for fluid translation.[9] It's still not perfect but as the system learns from the correct translation and the incorrect translation chances are it will be in the future.

Photo and video image data

Again the data being collected and stored is staggering. Digital cameras and smart phones are taking and sharing more photos and videos than ever before. Check out these stats:

- Each day 350 million photos are uploaded to Facebook, which equates to 4,000 photos per second.
- Flickr users upload 3.5 million photos to the site each day.
- Approximately 100 hours of video is uploaded to YouTube every minute.
- More than 45 million pictures are uploaded to Instagram every day.
- As of June 2013, Instagram users have shared more than 16 billion photos.[12]

Granted, sharing what we had for dinner or a picture of our new Labrador puppy won't change the world but this plethora of photo, video (and text data) is actually already saving lives in disaster areas.

When typhoon Hiayan hit the Philippines in 2013, for example, over 6,000 people were killed and 1.1 million homes were damaged or destroyed in hours. In the UK, a team of volunteers were creating a vital map of the damaged areas using just social media.

Because it is now very common for people to share their experiences as they happen in almost real time, photos, tweets (#Hiayan) and videos about the disaster were being posted on social media. In the aftermath of Hiayan the volunteers were receiving on average a million photos, messages, tweets, videos, etc., every day!
After filtering the millions of messages using artificial intelligence to pick out the ones that could be important the team of volunteers then made an assessment of what they saw. For example, for a photograph they would be asked, 'How much damage do you see?' and they simply needed to click the appropriate button: 'none', 'mild', or 'severe'. For text based messages such as tweets or Facebook updates the volunteer was asked to decide if the text was 'not relevant', 'request for help', 'infrastructure damage', 'population displacement', 'relevant but other', etc. Each piece of data (picture, video or message) was then assessed by between three to five different people to make sure the assessment was consistent and therefore probably accurate.
By pinpointing where the data was coming from in the Philippines (using GPS sensors in the photos or through the text) the work of the volunteers then created an online map, not just of the disaster zone but of the needs in each area.
That meant that when the disaster relief effort arrived in the Philippines they didn't need to waste days working out what was happening and where the worst hit areas were. They already knew from the map – created by people half way around the world – who needed water, who needed food, where the dead bodies were and where people had been displaced, where the most damage was and what hospitals were least damaged, and therefore more able to help the injured.[13]

How cool is that?

In addition to all the photo and video data created by individuals via their digital tech or smart phone there is also all the CCTV camera footage. In days gone by companies may video record their premises or retail store and store the recording for a week or so before recording over older recordings. Now some of the larger data savvy stores are keeping all the CCTV camera footage and analysing it to study how people walk through the shops, where they stop, what they look at and for how long so they can make alterations to offers and boost sales. Some are even using face recognition software so it probably won't be long before a combination of data sources such as CCTV camera footage, loyalty card information and face recognition software will see us being welcomed to a store on our smart phones and directed to particular special offers or promotions that may be of interest to us based on our previous buying habits!

Sensor data

There is also an increasing amount of data being generated and transmitted from sensors. There are sensors everywhere.

Have you ever wondered that makes your smart phone (or smart anything for that matter) smart? Basically what makes them smart is the inclusion of various sensors that capture data. In your smart phone for example there is a:

- GPS sensor
- Accelerometer sensor
- Gyroscope
- Proximity sensor
- Ambient sensor, and
- Near Field Communications (NFC) sensor.

The GPS (Global Positioning System) sensor lets us (and others) know where we are using the GPS satellite navigation system.
The GPS sensors in our phone can pinpoint our location within a few meters (assuming we are with our phone of course!). The accelerometer sensor is a motion sensor and measures the acceleration or how quickly the phone is moving. It's this technology that allows you to take better photos with your smart phone because it's this sensor that triggers the shutter when it detects the camera is stationary or stable. The gyroscope sensor is used to maintain orientation and is used to rotate the screen. It is this sensor that is often utilized in gaming apps where you have to tilt the screen to direct the character or steer the car. As the name would suggest the proximity sensor senses proximity and how close we are to other objects or locations. Ambient sensors are the ones that detect changes in the ambience or atmosphere so it is this sensor that adjusts the backlight on your phone or saves power when it's not being actively used. And finally the NFC sensor is one of the latest communication protocols being utilized in smart phones. It is these NFC sensors that when enabled, allow you to transfer funds just by bumping phones or waving your phone close to an appropriate payment machine.
There are also sensors in the natural environment, for example, in the oceans for measuring the health, temperature and changes of the oceans in real time. Also in Japan there are sensors in the soil to collect data on how healthy the soil is and companies are combining that data with weather data. Farmers can then subscribe to the service to get information to optimize yield, including how much and when to put fertilizer on their crops.
Increasingly more and more machines are equipped with sensors to monitor performance and provide information on when best to service or repair the machines.
For example, Rolls Royce manufactures nearly half the world's passenger jet engines including the Trent 1000, the engine that powers many of our transatlantic flights. When in operation these engines reach incredibly high temperatures – half the temperature of the surface of the sun and 200 degrees above that temperature when the metal used to

make the engine melts! The only reason it doesn't melt is because the engines are being cooled through special passageways and channels that keep the heat away from the metal.
Needless to say it's vital to know that everything is working and doing its job, as you wouldn't want the plane you are taking to visit your friends in New York to melt at 30,000 feet!
The engine is therefore full of vital components all engineered with absolute precision including an on-board computer that is the brains of the engine, controlling it and also collecting and monitoring data from sensors buried deep within the engine measuring 40 parameters 40 times per second including temperatures, pressures and turbine speeds.
All the measurements are stored in the computer and streamed via satellite back to Rolls Royce HQ in Derby, England. And that's true for the entire fleet of Rolls Royce engines, which is a lot of data when you consider that a Rolls Royce powered engine takes off or lands somewhere in the world every two and a half seconds.
Whenever those thousands of engines are in the air they are gathering data which is continuously sent back to HQ and constantly monitored using clever data analytics that are looking for anything unusual going on in the engine, or any sign that it may need to be serviced early or repaired. In Derby, computers then sift through the data to look for anomalies. If any are found they are immediately flagged and a human being will check the results and if necessary telephone the airline and work out what needs to be done – normally before the issue escalates into an actual problem.
These sensors therefore allow for dynamic maintenance based on actual engine-by-engine performance rather than some automatic rota system based on time alone. Instead of pulling an expensive piece of equipment out of service every three or six months these sensors allow the airlines to maintain their fleet much more cost effectively and, more importantly, these sensors make the planes much safer.[13]

Modern cars are also full of similar sensors that measure everything from fuel consumption to engine performance, which again allows for dynamic servicing and better long term performance.
On-board sensors also alert the driver if they get too close to another car or object and can even parallel park the car without the driver having to do anything!
In the retail industry, data has long been collected via barcode; however, the sensors known as Radio Frequency Identification (RFID) systems increasingly used by retailers and others are generating 100 to 1,000 times more data than the conventional barcode system.[14]
There are sensors everywhere.

The Internet of Things

The Internet of Things (IoT) is a result of more objects being manufactured with embedded sensors and the ability of those objects to communicate with each other.

IDC describes the IoT as:

> *'a network connecting – either wired or wireless – devices (things) that are characterized by automatic provisioning, management, and monitoring. It is innately analytical and integrated, and includes not just intelligent systems and devices, but connectivity enablement, platforms for device, network and application enablement, analytics and social business, and applications and vertical industry solutions. It is more than traditional machine-to-machine communication. Indeed, it is more than the traditional Information and Communications Technology (ICT) industry itself.'*[15]

This concept explores what is and will be possible as a result of advances in smart, sensor-based technology and massive advances in connectively between devices, systems and services that go way beyond business as usual. For example, research groups such as Gartner and ABI Research estimate that by 2020 there will be between 26 and 30 billion devices wirelessly connected to the IoT.

And the resulting information networks promise to create new business models and improve business processes and performance, while also reducing cost and potentially risk.

The day will come, not far from now when your alarm will be synced to your email account and if an early meeting is cancelled your alarm will automatically reset to a later time, which will also postpone the coffee machine to the new wake-up time. Your fridge will know what's in it and place online orders to replenish stocks without you having to do anything. You'll put on your suit, with a payment chip in the sleeve so you can swipe payment for lunch without a credit card.

Your wearable device or smart watch will monitor your health through the day, watching your calorie intake and making sure you stay active and don't sit too long at your desk. As you get in your car to drive home at night the car will automatically check the route with traffic and weather information to get you home as quickly and safely as possible. On arriving home, the temperature will be perfect and your fridge will tell you what you can make for dinner based on what you currently have in stock.

As you settle down to watch TV with your family, you may be enjoying a film rated 18 when your 5-year-old child walks in and your smart TV will suspend the film and change channel. Oh and if your elderly mother is ever house sitting while you are away your smart carpet will measure and monitor her movements and patterns – perhaps she goes to the kitchen at 10.30 a.m. every morning to make a cup of coffee or always goes to bed at 11 p.m. Should those patterns change you will be alerted to get in touch and check everything is OK.

The wired and wireless networks that connect the Internet of Things often use the same Internet Protocol (IP) that connects the Internet – hence the name. These vast networks create huge volumes of data that's then available for analysis. When objects use sensors to sense the environment and communicate with each other, they become tools for understanding complexity and responding to it quickly. The resulting physical information systems are now beginning to be deployed, and some of them operate without human intervention.

Pill-shaped micro-cameras already traverse the human digestive tract and send back thousands of images to pinpoint sources of illness. Precision farming equipment with wireless links to data collected from remote satellites and ground sensors can take into account crop conditions and adjust the way each individual part of a field is farmed. There are even billboards in Japan that monitor passers-by, assess how they fit consumer profiles, and instantly change displayed messages based on those assessments. Advances in wireless networking technology and the greater standardization of communication protocols make it possible to collect data from these sensors almost anywhere at any time. Ever-smaller silicon chips are gaining new capabilities, while costs are falling. Massive increases in storage and computing power, some of it available via cloud computing, make number crunching possible on a very large scale and at declining cost.[16]

All coming together to create Big Data.

1.8.2 The anatomy of Big Data [8]

When we consider the types and forms of data that now exists it's easy to see how people become overwhelmed and bamboozled by the possibilities of Big Data. Although, as I've said I think the term will disappear and what we consider Big Data today will just be 'data' tomorrow.

For a start, what is uncommon and exciting now will become commonplace. Plus the term may be simple and easy to remember but it's overly simplistic and places far too much emphasis on the volume of data. But volume is just one of the four V's of Big Data:

- ***Volume*** – relating to the vast amounts of data generated every second.
- ***Velocity*** – relating to the speed at which new data is generated and moves around the world. For example, credit card fraud detection tracks millions of transactions for unusual patterns in almost real time.
- ***Variety*** – relating to the increasingly different types of data that is being generated from financial data to social media feeds; from photos to sensor data; from video footage to voice recordings.
- ***Veracity*** – relating to the messiness of the data being generated – just think of Twitter posts with hash tags, abbreviations, typos, text language and colloquial speech.

Big Data backlash

As with any new frontier, the frontier of Big Data is also under attack. There are those that believe that it's a storm in a teacup and the theory of Big Data is so far removed from the reality for most businesses that it will never yield much, if any fruit for the vast majority of business.

Certainly there are some companies that already have these huge data sets; however, most businesses will never have access to the volume and variety of data that an Amazon, eBay or Facebook will have. But as I've said before that's OK because most businesses don't need access to oceans of data.

The other area of attack is around consumer data and privacy.

The reputation of Big Data has suffered with the revelations by whistleblower Edward Snowden that the US National Security Agency (NSA) has been systematically using Big Data analytics to 'spy' on everyone's communications as well as perform targeted surveillance of individuals and companies. We can all be certain that the US is not the only government agency in the world to collect and use Big Data. For example, former French foreign minister, Bernard Kouchner, stated, 'Let's be honest, we eavesdrop too.

Everyone is listening to everyone else. But we don't have the same means as the United States, which makes us jealous.'

Despite high profile Snowden-type media stories, most people are completely unaware of just how much data about them is freely available online. Even if someone takes the time to complete privacy settings on social media and is deliberately vague and cautious about over-sharing – there is still a phenomenal amount of information being collected, stored and analysed. Most of us are, for example, almost entirely oblivious to the fact that the GPS sensor in their smart phone makes it possible to identify where a picture was taken within a few meters, regardless of whether the person sharing the photo adds a tag, message or caption. They don't realize how open and freely available their social media sites are, how much of what they post is saved and analysed – even when the platform tells its users that the photo or video will self-destruct in 10 seconds! Those images may not be accessible to the user after a set time but they are saved. They have no idea that their web browser is monitoring their every move or even that people can easily hack into the camera on their laptop and watch them!

In 2013 a 19-year-old US student was charged with hacking Miss Teen USA's webcam. The FBI found that he had used malicious software to remotely operate webcams to get nude photos and videos of at least seven women as they changed clothes. Some of these women he knew personally and others he found by hacking Facebook pages. In the UK in 2014 another man received a suspended sentence for the same thing. Probably best to cover your webcam when you're not using it – just in case!

So far people have not really cottoned on to the dangers or the inherent value of their own data and are happy to freely share that data in exchange for services they want, such as Facebook.

Facebook is already a gigantic data mining paradise with unbelievable amounts of data at their disposal, all enthusiastically provided by the users of Facebook. Remember the stats from earlier – 350 million photos a day, 293,000 status updates a minute and 25% of users never bother with privacy!

Facebook knows what we look like, who our friends are, what our views are, what our interests are, when our birthday is, whether we are in a relationship or not, where we are, what we like and dislike, and much more. That is an awful lot of information (and power) in the hands of one commercial company.

People may start to get uncomfortable about the amount of data that is known and held amount them. But how much of a difference would it really make? Take Facebook again: even if we all stopped using Facebook today (which is very unlikely), the company would still have more information about people than any other private company on the planet. Google may come close but they don't have the plethora of detailed personal data that Facebook has. Of course it's not just Facebook.

The challenge is that once companies have access to the data they won't stop. And we don't have to be a loyalty card member for the companies to know about us: in addition to social media, they can also track our credit card use and use face recognition software to record what we are doing in store.

A recent study showed that it is possible to accurately predict a range of highly sensitive personal attributes simply by analyzing the 'Likes' we have clicked on Facebook. The work conducted by researchers at Cambridge University and Microsoft Research shows how the patterns of Facebook 'Likes' can very accurately predict characteristics such as your sexual orientation, satisfaction with life, intelligence, emotional stability, religion, alcohol use and drug use, relationship status, age, gender, race and political views among many others.[17]

The fact is that the data collectively held on you by banks, credit card companies, insurance companies, supermarkets and social media is astonishing and it's growing all the time.

Even if people did become uncomfortable in enough numbers to bring about changes to legislation it may be too late. It would be like shutting the barn door once the horse had bolted. It may be that legislation may push for at least some of the most sensitive data to be anonymized, i.e. markers that identify an actual person to be removed, but it will still be used and the datifaction of the world will not stop.

Whether we like it or not, or are ready for it or not, the future will involve Big Data. Our ability to harness that power with intelligence, common sense and practicality will see us turn it into meaningful SMART Data.

Having started with strategy and identified the SMART questions around customers, finance, operations, resources and risk you need to figure out what metrics and data you actually need access to in order to answer those questions, which in turn will help you to deliver your strategy.

1.9. BIG DATA TOOLS AND TECHNIQUES [18]

1.9.1 Understanding Big Data Storage [18]

As we have discussed in much of the book so far, most, if not all big data applications achieve their performance and scalability through deployment on a collection of storage and computing resources bound together within a runtime environment. In essence, the ability to design, develop, and implement a big data application is directly dependent on an awareness of the architecture of the underlying computing platform, both from a hardware and more importantly from a software perspective.
One commonality among the different appliances and frameworks is the adaptation of tools to leverage the combination of collections of four key computing resources:

1. ***Processing capability*** often referred to as a CPU, processor, or node. Generally speaking, modern processing nodes often incorporate multiple cores that are individual CPUs that share the node's memory and are managed and scheduled together, allowing multiple tasks to be run simultaneously; this is known as multithreading.
2. ***Memory***, which holds the data that the processing node is currently working on. Most single node machines have a limit to the amount of memory.
3. ***Storage***, providing persistence of data—the place where datasets are loaded, and from which the data is loaded into memory to be processed.
4. ***Network***, which provides the "pipes" through which datasets are exchanged between different processing and storage nodes.

Because single-node computers are limited in their capacity, they cannot easily accommodate massive amounts of data. That is why the high-performance platforms are composed of collections of computers in which the massive amounts of data and requirements for processing can be distributed among a pool of resources.

1.9.2 A Generals Overview of High-Performance Architecture [18]

Most high-performance platforms are created by connecting multiple nodes together via a variety of network topologies. Specialty appliances may differ in the specifics of the configurations, as do software appliances. However, the general architecture distinguishes the management of computing resources (and corresponding allocation of

tasks) and the management of the data across the network of storage nodes, as is seen in Figure 1.12.

In this configuration, a master job manager oversees the pool of processing nodes, assigns tasks, and monitors the activity. At the same time, a storage manager oversees the data storage pool and distributes datasets across the collection of storage resources. While there is no a priori requirement that there be any colocation of data and processing tasks, it is beneficial from a performance perspective to ensure that the threads process data that is local, or close to minimize the costs of data access latency.

To get a better understanding of the layering and interactions within a big data platform, we will examine the Apache Hadoop software stack, since the architecture is published and open for review.

Hadoop is essentially a collection of open source projects that are combined to enable a software-based big data appliance. We begin with the core aspects of Hadoop's utilities, upon which the next layer in the stack is propped, namely Hadoop distributed file systems (HDFS) and MapReduce. A new generation framework for job scheduling and cluster management is being developed under the name YARN.

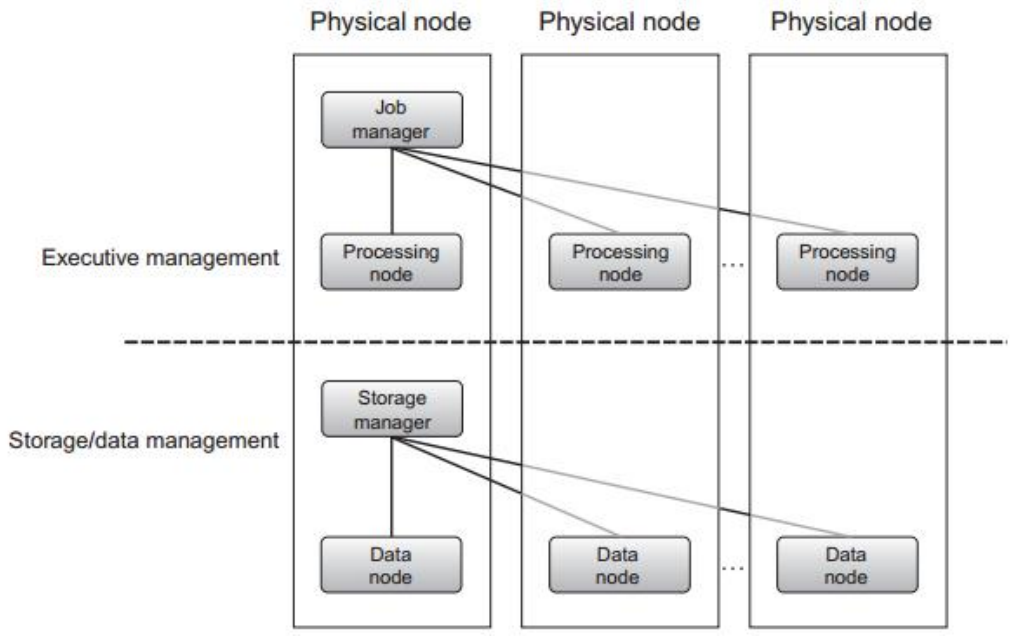


Figure 1.12 Typical organization of resources in a big data platform [18]

1.9.3 HDFS [18]

HDFS attempts to enable the storage of large files, and does this by distributing the data among a pool of data nodes. A single name node (sometimes referred to as NameNode) runs in a cluster, associated with one or more data nodes, and provide the management of a typical hierarchical file organization and namespace. The name node effectively coordinates the interaction with the distributed data nodes.

The creation of a file in HDFS appears to be a single file, even though it blocks "chunks" of the file into pieces that are stored on individual data nodes.

The name node maintains metadata about each file as well as the history of changes to file metadata. That metadata includes an enumeration of the managed files, properties of the files, and the file system, as well as the mapping of blocks to files at the data nodes.

The data node itself does not manage any information about the logical HDFS file; rather, it treats each data block as a separate file and shares the critical information with the name node.

Once a file is created, as data is written to the file, it is actually cached in a temporary file. When the amount of the data in that temporary file is enough to fill a block in an HDFS file, the name node is alerted to transition that temporary file into a block that is committed to a permanent data node, which is also then incorporated into the file management scheme.

HDFS provides a level of fault tolerance through data replication. An application can specify the degree of replication (i.e., the number of copies made) when a file is created. The name node also manages replication, attempting to optimize the marshaling and communication of replicated data in relation to the cluster's configuration and corresponding efficient use of network bandwidth. This is increasingly important in larger environments consisting of multiple racks of data servers, since communication among nodes on the same rack is generally faster than between server node sin different racks. HDFS attempts to maintain awareness of data node locations across the hierarchical configuration.

In essence, HDFS provides performance through distribution of data and fault tolerance through replication. The result is a level of robustness for reliable massive file storage. Enabling this level of reliability should be facilitated through a number of key tasks for failure management, some of which are already deployed within HDFS while others are not currently implemented:

• ***Monitoring***: There is a continuous "heartbeat" communication between the data nodes to the name node. If a data node's heartbeat is not heard by the name node, the data node is considered to have failed and is no longer available. In this case, a replica is employed to replace the failed node, and a change is made to the replication scheme.

• ***Rebalancing***: This is a process of automatically migrating blocks of data from one data node to another when there is free space, when there is an increased demand for the data and moving it may improve performance (such as moving from a traditional disk drive to a solid-state drive that is much faster or can accommodate increased numbers of simultaneous accesses), or an increased need to replication in reaction to more frequent node failures.

• ***Managing integrity***: HDFS uses checksums, which are effectively "digital signatures", associated with the actual data stored in a file (often calculated as a numerical function of

the values within the bits of the files) that can be used to verify that the data stored corresponds to the data shared or received. When the checksum calculated for a retrieved block does not equal the stored checksum of that block, it is considered an integrity error. In that case, the requested block will need to be retrieved from a replica instead.

• ***Metadata replication***: The metadata files are also subject to failure, and HDFS can be configured to maintain replicas of the corresponding metadata files to protect against corruption.

• ***Snapshots***: This is incremental copying of data to establish a point in time to which the system can be rolled back.[19, 20]

These concepts map to specific internal protocols and services that HDFS uses to enable a large-scale data management file system that can run on commodity hardware components. The ability to use HDFS solely as a means for creating a scalable and expandable file system for maintaining rapid access to large datasets provides a reasonable value proposition from an Information Technology perspective:

• decreasing the cost of specialty large-scale storage systems;
• providing the ability to rely on commodity components;
• enabling the ability to deploy using cloud-based services;
• reducing system management costs.

1.9.4 MapReduce and YARN [18]

In Hadoop, MapReduce originally combined both job management and oversight and the programming model for execution. The MapReduce execution environment employs a master/slave execution model, in which one master node (called the JobTracker) manages a pool of slave computing resources (called TaskTrackers) that arecalled upon to do the actual work. The role of the JobTracker is to manage the resources with some specific responsibilities, including managing the TaskTrackers, continually monitoring their accessibility and availability, and the different aspects of job management that include scheduling tasks, tracking the progress of assigned tasks, reacting to identified failures, and ensuring fault tolerance of the execution. The role of the TaskTracker is much simpler: wait for a task assignment, initiate and execute the requested task, and provide status back to the JobTracker on a periodic basis. Different clients can make requests from the JobTracker, which becomes the sole arbitrator for allocation of resources.

There are limitations within this existing MapReduce model. First, the programming paradigm is nicely suited to applications where there is locality between the processing and the data, but applications that demand data movement will rapidly become bogged down by network latency issues. Second, not all applications are easily mapped to the MapReduce model, yet applications developed using alternative programming methods

would still need the MapReduce system for job management. Third, the allocation of processing nodes within the cluster is fixed through allocation of certain nodes as "map slots" versus "reduce slots." When the computation is weighted toward one of the phases, the nodes assigned to the other phase are largely unused, resulting in processor underutilization.

This is being addressed in future versions of Hadoop through the segregation of duties within a revision called YARN. In this approach, overall resource management has been centralized while management of resources at each node is now performed by a local NodeManager. In addition, there is the concept of an ApplicationMaster that is associated with each application that directly negotiates with the central ResourceManager for resources while taking over the responsibility for monitoring progress and tracking status. Pushing this responsibility to the application environment allows greater flexibility in the assignment of resources as well as be more effective in scheduling to improve node utilization.

Last, the YARN approach allows applications to be better aware of the data allocation across the topology of the resources within a cluster. This awareness allows for improved colocation of compute and data resources, reducing data motion, and consequently, reducing delays associated with data access latencies. The result should be increased scalability and performance.[21]

1.9.5 Expanding the Big Data Application Ecosystem [18]

At this point, a few key points regarding the development of big data applications should be clarified. First, despite the simplicity of downloading and installing the core components of a big data development and execution environment like Hadoop, designing, developing, and deploying analytic applications still requires some skill and expertise. Second, one must differentiate between the tasks associated with application design and development and the tasks associated with architecting the big data system, selecting and connecting its components, system configuration, as well as system monitoring and continued maintenance.

In other words, transitioning from an experimental "laboratory" system into a production environment demands more than just access to the computing, memory, storage, and network resources. There is a need to expand the ecosystem to incorporate a variety of additional capabilities, such as configuration management, data organization, application development, and optimization, as well as additional capabilities to support analytical processing. Our examination of a prototypical big data platform engineered using Hadoop continues by looking at a number of additional components that might typically be considered as part of the ecosystem.

1.9.6 ZOOKEEPER [18]

Whenever there are multiple tasks and jobs running within a single distributed environment, there is a need for configuration management and synchronization of various aspects of naming and coordination. The project's web page specifies it more clearly: "Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services."[22]
Zookeeper manages a naming registry and effectively implements a system for managing the various static and ephemeral named objects in a hierarchical manner, much like a file system. In addition, it enables coordination for exercising control over shared resources that are impacted by race conditions (in which the expected output of a process is impacted by variations in timing) and deadlock (in which multiple tasks vying for control of the same resource effectively lock each other out of any task's ability to use the resource). Shared coordination services like those provided in Zookeeper allow developers to employ these controls without having to develop them from scratch.

1.9.7 HBASE [18]

HBase is another example of a nonrelational data management environment that distributes massive datasets over the underlying Hadoop framework. HBase is derived from Google's BigTable and is a column-oriented data layout that, when layered on top of Hadoop, provides a fault-tolerant method for storing and manipulating large data tables. Data stored in a columnar layout is amenable to compression, which increases the amount of data that can be represented while decreasing the actual storage footprint. In addition, HBase supports in-memory execution.
HBase is not a relational database, and it does not support SQL queries. There are some basic operations for HBase: **Get** (which access a specific row in the table), **Put** (which stores or updates a row in the table), **Scan** (which iterates over a collection of rows in the table), and **Delete** (which removes a row from the table). Because it can be used to organize datasets, coupled with the performance provided by the aspects of the columnar orientation, HBase is a reasonable alternative as a persistent storage paradigm when running MapReduce applications.

1.9.8 HIVE [18]

One of the often-noted issues with MapReduce is that although it provides a methodology for developing and executing applications that use massive amounts of data, it is not more than that. And while the data can be managed within files using HDFS, many business applications expect representations of data in structured database tables. That

was the motivation for the development of Hive, which (according to the Apache Hive web site[23]) is a "data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems." Hive is specifically engineered for data warehouse querying and reporting and is not intended for use as within transaction processing systems that require real-time query execution or transaction semantics for consistency at the row level.

Hive is layered on top of the file system and execution framework for Hadoop and enables applications and users to organize data in a structured data warehouse and therefore query the data using a query language called HiveQL that is similar to SQL (the standard Structured Query Language used for most modern relational database management systems). The Hive system provides tools for extracting/transforming/loading data (ETL) into a variety of different data formats. And because the data warehouse system is built on top of Hadoop, it enables native access to the MapReduce model, allowing programmers to develop custom Map and Reduce functions that can be directly integrated into HiveQL queries. Hive provides scalability and extensibility for batch-style queries for reporting over large datasets that are typically being expanded while relying on the fault tolerant aspects of the underlying Hadoop execution model.

1.9.9 PIG [18]

Even though the MapReduce programming model is relatively straightforward, it still takes some skill and understanding of both parallel and distributed programming and Java to best take advantage of the model. The Pig project is an attempt at simplifying the application development process by abstracting some of the details away through a higher level programming language called Pig Latin. According to the project's web site[24], Pig's high-level programming language allows the developer to specify how the analysis is performed. In turn, a compiler transforms the Pig Latin specification into MapReduce programs.

The intent is to embed a significant set of parallel operators and functions contained within a control sequence of directives to be applied to datasets in a way that is somewhat similar to the way SQL statements are applied to traditional structured databases. Some examples include generating datasets, filtering out subsets, joins, splitting datasets, removing duplicates. For simple applications, using Pig provides significant ease of development, and more complex tasks can be engineered as sequences of applied operators.

In addition, the use of a high-level language also allows the compiler to identify opportunities for optimization that might have been ignored by an inexperienced

programmer. At the same time, the Pig environment allows developers to create new user defined functions (UDFs) that can subsequently be incorporated into developed programs.

1.9.10 MAHOUT [18]

Attempting to use big data for analytics would be limited without any analytics capabilities. Mahout is a project to provide a library of scalable implementations of machine learning algorithms on top of MapReduce and Hadoop. As is described at the project's home page[25],
Mahout's library includes numerous well-known analysis methods including:

• ***Collaborative filtering and other user and item-based recommender algorithms***, which is used to make predictions about an individual's interest or preferences through comparison with a multitude of others that may or may not share similar characteristics.
• ***Clustering***, including K-Means, Fuzzy K-Means, Mean Shift, and Dirichlet process clustering algorithms to look for groups, patterns, and commonality among selected cohorts in a population.
• ***Categorization*** using Naïve Bayes or decision forests to place items into already defined categories.
• ***Text mining*** and topic modeling algorithms for scanning text and assigning contextual meanings.
• ***Frequent pattern mining***, which is used for market basket analysis, comparative health analytics, and other patterns of correlation within large datasets.

Mahout also supports other methods and algorithms. The availability of implemented libraries for these types of analytics free the development team to consider the types of problems to be analyzed and more specifically, the types of analytical models that can be applied to seek the best answers.

Variable	Intent	Technical Requirements
Predisposition to parallelization	Number and type of processing node(s)	Number or processors Types of processors
Size of data to be persistently stored	Amount and allocation of disk space for distributed file system	Size of disk drives
		Number of disk drives
		Type of drives (SSD versus magnetic versus optical)
		Bus configuration (shared everything versus shared nothing, for example)
Amount of data to be accessible in memory	Amount and allocation of core memory	Amount of RAM memory Cache memories
Need for cross-node communication	Optimize speed and bandwidth	Network/cabinet configuration
		Network speed
		Network bandwidth
Types of data organization	Data management requirements	File management organization
		Database requirements
		Data orientation (row versus column)
		Type of data structures
Developer skill set	Development tools	Types of programming tools, compilers, execution models, debuggers, etc.
Types of algorithms	Analytic functionality requirements	Data warehouse/marts for OLAP Data mining and predictive analytics

Table 1.9 Variables to Consider When Framing Big Data Environment [18]

1.9.11 Considerations [18]

Big data analytics applications employ a variety of tools and techniques for implementation. When organizing your thoughts about developing those applications, it is important to think about the parameters that will frame your needs for technology evaluation and acquisition, sizing and configuration, methods of data organization, and required algorithms to be used or developed from scratch.

Prior to diving directly into downloading and installing software, focus on the types of big data business applications and their corresponding performance scaling needs, such as those listed in Table 1.9.

The technical requirements will guide both the hardware and the software configurations. This also allows you to align the development of the platform with the business application development needs.

CHAPTER 2

BIG DATA PLATFORMS - HADOOP

2.1 A BRIEF HISTORY OF HADOOP [31]

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

The Origin of the Name "Hadoop"

The name Hadoop is not an acronym; it's a made-up name.The project's creator, Doug Cutting, explains how the name came about:

The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.

Sub projects and "contrib" modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the job racker keeps track of MapReduce jobs.

Building a web search engine from scratch was an ambitious goal, for not only is the software required to crawl and index websites complex to write, but it is also a challenge to run without a dedicated operations team, since there are so many moving parts. It's expensive, too: Mike Cafarella and Doug Cutting estimated a system supporting a 1-billion-page index would cost around half a million dollars in hardware, with a monthly running cost of $30,000.§ Nevertheless, they believed it was a worthy goal, as it would open up and ultimately democratize search engine algorithms.

Nutch was started in 2002, and a working crawler and search system quickly emerged. However, they realized that their architecture wouldn't scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google's distributed filesystem, called GFS, which was being used in production at Google.GFS, or something like it, would solve their storage needs for the very large files generated as a part of the web crawl and indexing process. In particular, GFS would free up time being spent on administrative tasks such as managing storage

nodes. In 2004, they set about writing an open source implementation, the Nutch Distributed Filesystem (NDFS).

In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS.

NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see sidebar). This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster.

In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community.By this time, Hadoop was being used by many other companies besides Yahoo!,such as Last.fm, Facebook, and the New York Times. Some applications are covered in the case studies in Chapter 16 and on the Hadoop wiki.

In one well-publicized feat, the New York Times used Amazon's EC2 compute cloud to crunch through four terabytes of scanned archives from the paper converting them to PDFs for the Web. The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked on without the combination of Amazon's pay-by-the-hour model (which allowed the NYT to access a large number of machines for a short period) and Hadoop's easy-to-use parallel programming model.

In April 2008, Hadoop broke a world record to become the fastest system to sort a terabyte of data. Running on a 910-node cluster, Hadoop sorted one terabyte in 209 seconds (just under 3½ minutes), beating the previous year's winner of 297 seconds (described in detail in "TeraByte Sort on Apache Hadoop"). In November of the same year, Google reported that its MapReduce implementation sorted one terabyte in 68 seconds. As the first edition of this book was going to press (May 2009), it was announced that a team at Yahoo! used Hadoop to sort one terabyte in 62 seconds.

2.2 BIG DATA & HADOOP – RESTAURANT ANALOGY [32]

Let us take an analogy of a restaurant to understand the problems associated with Big Data and how Hadoop solved that problem.

Bob is a businessman who has opened a small restaurant. Initially, in his restaurant, he used to receive two orders per hour and he had one chef with one food shelf in his restaurant which was sufficient enough to handle all the orders.

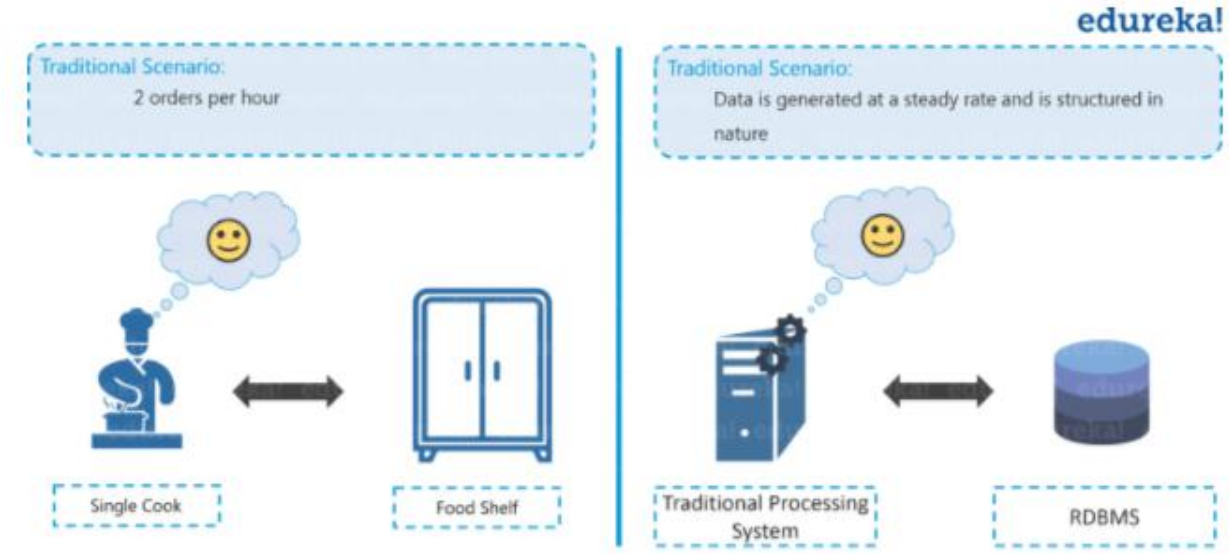


Figure 2.1 Traditional Restaurant Scenario [32]

Now let us compare the restaurant example with the traditional scenario where data was getting generated at a steady rate and our traditional systems like RDBMS is capable enough to handle it, just like Bob's chef. Here, you can relate the data storage with the restaurant's food shelf and the traditional processing unit with the chef as shown in the figure above.

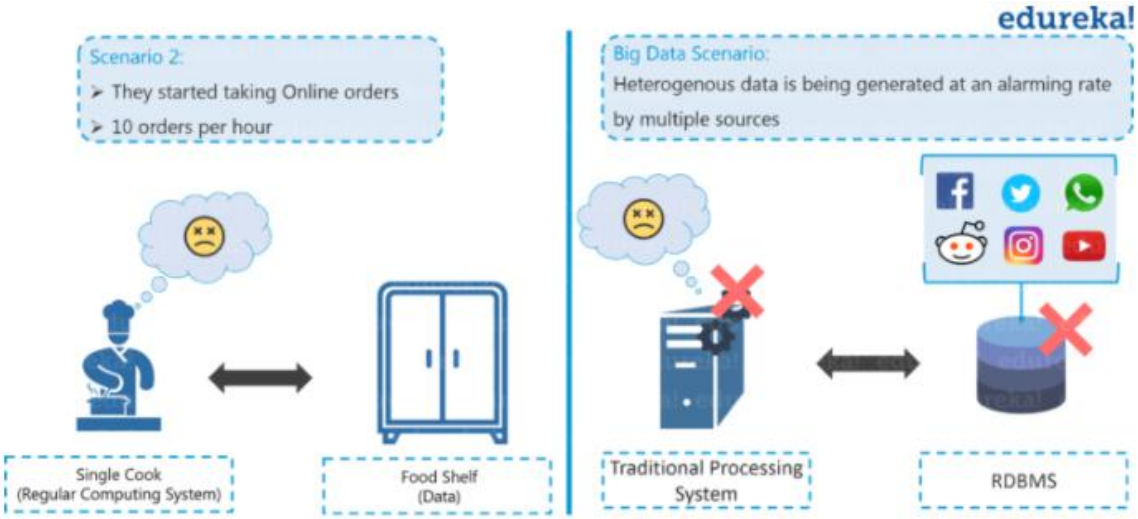


Figure 2.2 Traditional Scenario [32]

After few months, Bob thought of expanding his business and therefore, he started taking online orders and added few more cuisines to the restaurant's menu in order to engage a larger audience. Because of this transition, the rate at which they were receiving orders rose to an alarming figure of 10 orders per hour and it became quite difficult for a single cook to cope up with the current situation. Aware of the situation in processing the orders, Bob started thinking about the solution.

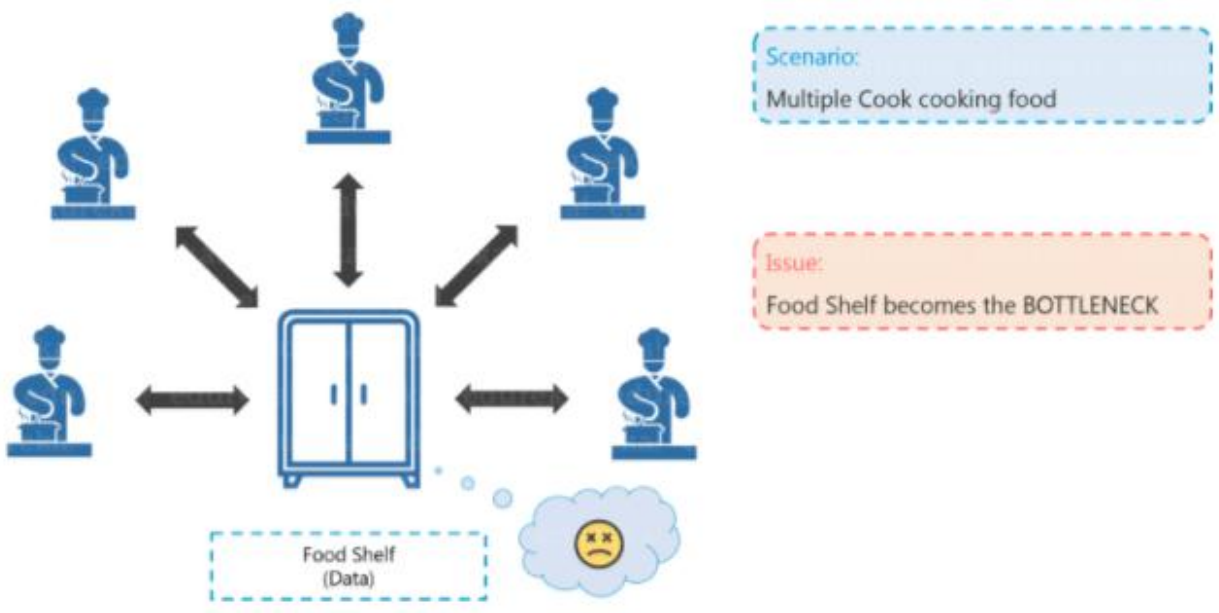


Figure 2.3 Distributed Processing Scenario [32]

Similarly, in Big Data scenario, the data started getting generated at an alarming rate because of the introduction of various data growth drivers such as social media, smartphones etc. Now, the traditional system, just like cook in Bob's restaurant, was not efficient enough to handle this sudden change. Thus, there was a need for a different kind of solutions strategy to cope up with this problem.

After a lot of research, Bob came up with a solution where he hired 4 more chefs to tackle the huge rate of orders being received. Everything was going quite well, but this solution led to one more problem. Since four chefs were sharing the same food shelf, the very food shelf was becoming the bottleneck of the whole process. Hence, the solution was not that efficient as Bob thought.

Figure 2.4 Distributed Processing Scenario Failure [32]

Similarly, to tackle the problem of processing huge datasets, multiple processing units were installed so as to process the data parallelly (just like Bob hired 4 chefs). But even in this case, bringing multiple processing units was not an effective solution because: the centralized storage unit became the bottleneck. In other words, the performance of the whole system is driven by the performance of the central storage unit. Therefore, the moment our central storage goes down, the whole system gets compromised. Hence, again there was a need to resolve this single point of failure.

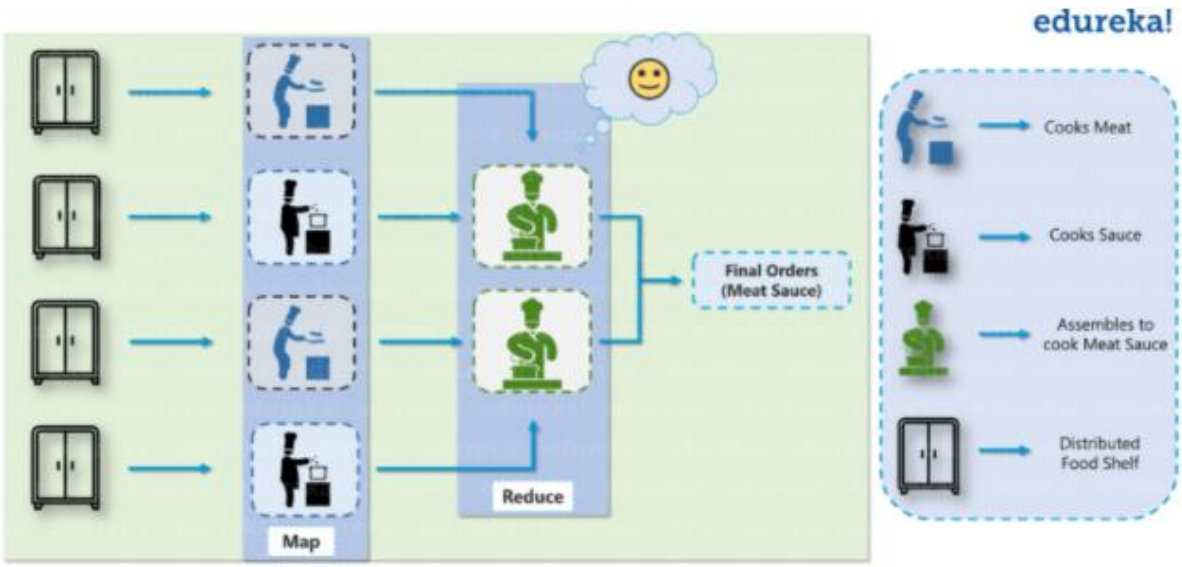


Figure 2.5 Solution to Restaurant Problem [32]

Bob came up with another efficient solution, he divided all the chefs in two hierarchies, i.e. junior and head chef and assigned each junior chef with a food shelf. Let us assume that the dish is Meat Sauce. Now, according to Bob's plan, one junior chef will prepare meat and the other junior chef will prepare the sauce. Moving ahead they will transfer both meat and sauce to the head chef, where the head chef will prepare the meat sauce after combining both the ingredients, which then will be delivered as the final order.

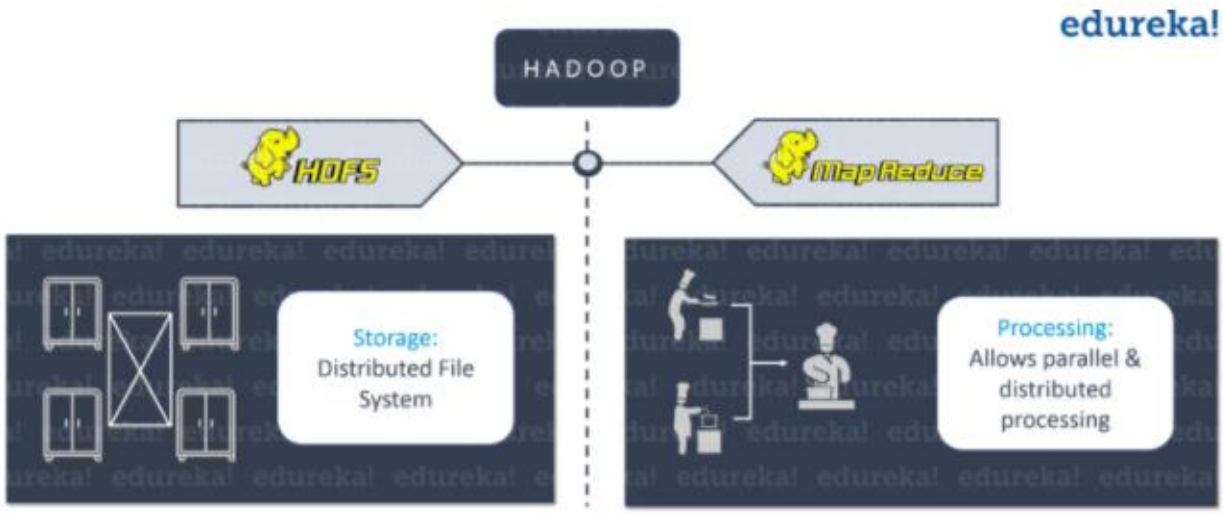


Figure 2.6 Hadoop in Restaurant Analogy [32]

Hadoop functions in a similar fashion as Bob's restaurant. As the food shelf is distributed in Bob's restaurant, similarly, in Hadoop, the data is stored in a distributed fashion with replications, to provide fault tolerance. For parallel processing, first the data is processed by the slaves where it is stored for some intermediate results and then those intermediate results are merged by master node to send the final result.

Now, you must have got an idea why Big Data is a problem statement and how Hadoop solves it. As we just discussed above, there were three major challenges with Big Data:

- ***The first problem is storing the colossal amount of data***. Storing huge data in a traditional system is not possible. The reason is obvious, the storage will be limited to one system and the data is increasing at a tremendous rate.
- ***The second problem is storing heterogeneous data***. Now we know that storing is a problem, but let me tell you it is just one part of the problem. The data is not only huge, but it is also present in various formats i.e. unstructured, semi-structured and structured. So, you need to make sure that you have a system to store different types of data that is generated from various sources.

- ***Finally let's focus on the third problem, which is the processing speed***. Now the time taken to process this huge amount of data is quite high as the data to be processed is too large.

To solve the storage issue and processing issue, two core components were created in Hadoop – HDFS and YARN. HDFS solves the storage issue as it stores the data in a distributed fashion and is easily scalable. And, YARN solves the processing issue by reducing the processing time drastically. Moving ahead, let us understand what is Hadoop?

2.3 WHAT IS HADOOP? [32]

Hadoop is an open-source software framework used for storing and processing Big Data in a distributed manner on large clusters of commodity hardware. Hadoop is licensed under the Apache v2 license. Hadoop was developed, based on the paper written by Google on MapReduce system and it applies concepts of functional programming. Hadoop is written in the Java programming language and ranks among the highest-level Apache projects. Hadoop was developed by Doug Cutting and Michael J. Cafarella.

2.3.1 Hadoop-as-a-Solution [32]

Let's understand how Hadoop provides solution to the Big Data problems that we have discussed so far.

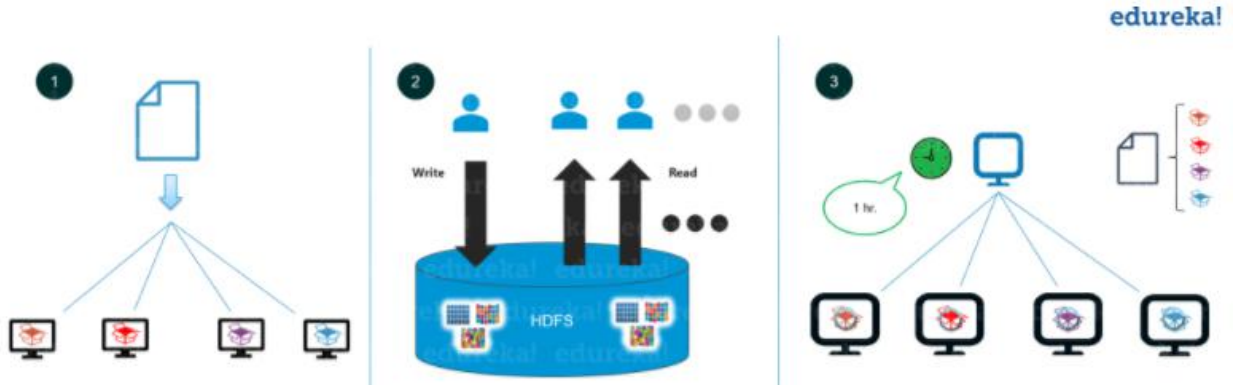


Figure 2.7 Hadoop-as-a-Solution [32]

The first problem is storing huge amount of data.

As you can see in the above image, HDFS provides a distributed way to store Big Data. Your data is stored in blocks in DataNodes and you specify the size of each block. Suppose you have 512MB of data and you have configured HDFS such that it will create 128 MB of data blocks. Now, HDFS will divide data into 4 blocks as 512/128=4 and

stores it across different DataNodes. While storing these data blocks into DataNodes, data blocks are replicated on different DataNodes to provide fault tolerance.

Hadoop follows horizontal scaling instead of vertical scaling. In horizontal scaling, you can add new nodes to HDFS cluster on the run as per requirement, instead of increasing the hardware stack present in each node.

Next problem was storing the variety of data.
As you can see in the above image, in HDFS you can store all kinds of data whether it is structured, semi-structured or unstructured. In HDFS, there is no pre-dumping schema validation. It also follows write once and read many model. Due to this, you can just write any kind of data once and you can read it multiple times for finding insights.

The third challenge was about processing the data faster.
In order to solve this, we move processing unit to data instead of moving data to processing unit. So, what does it mean by moving the computation unit to data? It means that instead of moving data from different nodes to a single master node for processing, the processing logic is sent to the nodes where data is stored so as that each node can process a part of data in parallel. Finally, all of the intermediary output produced by each node is merged together and the final response is sent back to the client.

2.3.2 Hadoop Features [32]

Reliability:
When machines are working in tandem, if one of the machines fails, another machine will take over the responsibility and work in a reliable and fault tolerant fashion. Hadoop infrastructure has inbuilt fault tolerance features and hence, Hadoop is highly reliable.

Economical:
Hadoop uses commodity hardware (like your PC, laptop). For example, in a small Hadoop cluster, all your DataNodes can have normal configurations like 8-16 GB RAM with 5-10 TB hard disk and Xeon processors, but if I would have used hardware-based RAID with Oracle for the same purpose, I would end up spending 5x times more at least. So, the cost of ownership of a Hadoop-based project is pretty minimized. It is easier to maintain the Hadoop environment and is economical as well. Also, Hadoop is an open source software and hence there is no licensing cost.

Scalability:
Hadoop has the inbuilt capability of integrating seamlessly with cloud-based services. So, if you are installing Hadoop on a cloud, you don't need to worry about the scalability

factor because you can go ahead and procure more hardware and expand your setup within minutes whenever required.

Flexibility:
Hadoop is very flexible in terms of ability to deal with all kinds of data. We discussed "Variety" in our previous blog on Big Data Tutorial, where data can be of any kind and Hadoop can store and process them all, whether it is structured, semi-structured or unstructured data.

These 4 characteristics make Hadoop a front-runner as a solution to Big Data challenges. Now that we know what is Hadoop, we can explore the core components of Hadoop. Let us understand, what are the core components of Hadoop.

2.3.3 Hadoop Core Components [32]

While setting up a Hadoop cluster, you have an option of choosing a lot of services as part of your Hadoop platform, but there are two services which are always mandatory for setting up Hadoop. One is HDFS (storage) and the other is YARN (processing). HDFS stands for Hadoop Distributed File System, which is a scalable storage unit of Hadoop whereas YARN is used to process the data i.e. stored in the HDFS in a distributed and parallel fashion.

HDFS
Let us go ahead with HDFS first. The main components of HDFS are: NameNode and DataNode. Let us talk about the roles of these two components in detail.

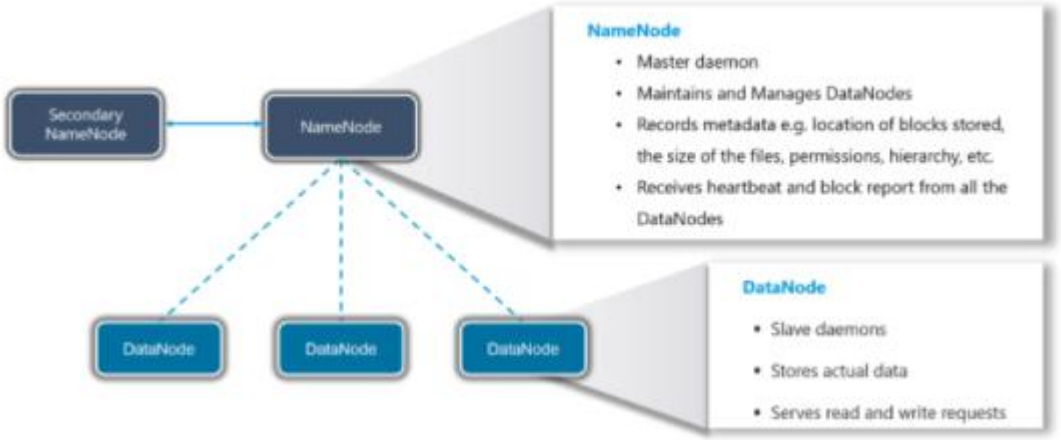


Figure 2.8 HDFS [32]

NameNode

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the blocks stored in the cluster, e.g. location of blocks stored, size of the files, permissions, hierarchy, etc.
- It records each and every change that takes place to the file system metadata
- If a file is deleted in HDFS, the NameNode will immediately record this in the EditLog
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live
- It keeps a record of all the blocks in the HDFS and DataNode in which they are stored
- It has high availability and federation features which I will discuss in HDFS architecture in detail

DataNode

- It is the slave daemon which run on each slave machine
- The actual data is stored on DataNodes
- It is responsible for serving read and write requests from the clients
- It is also responsible for creating blocks, deleting blocks and replicating the same based on the decisions taken by the NameNode
- It sends heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds

So, this was all about HDFS in nutshell. Now, let move ahead to our second fundamental unit of Hadoop i.e. YARN.

YARN

YARN comprises of two major component: ResourceManager and NodeManager.

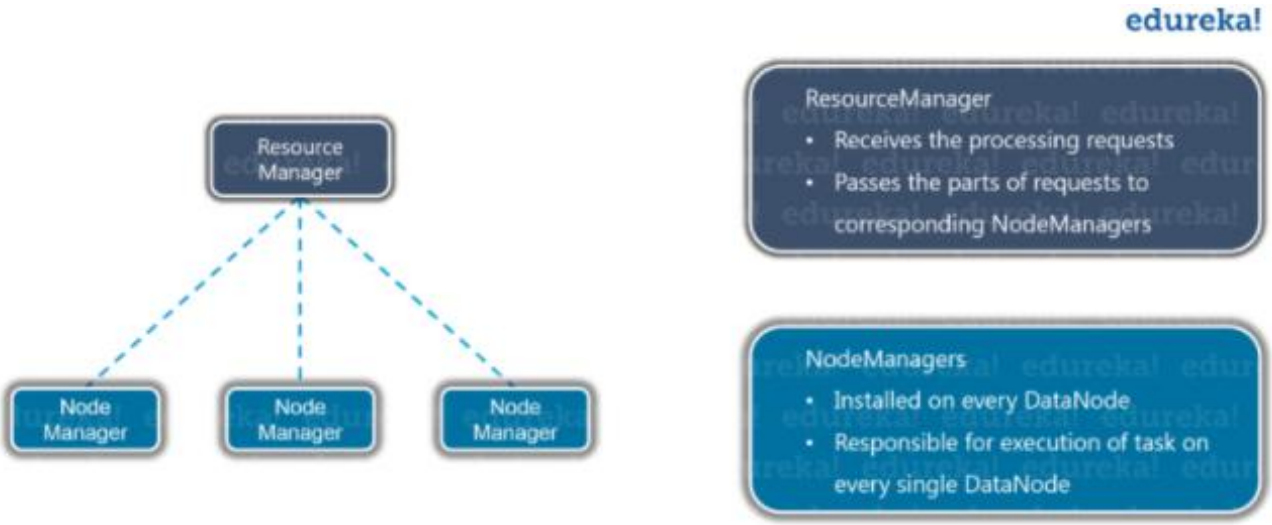


Figure 2.9 YARN [32]

ResourceManager

- It is a cluster level (one for each cluster) component and runs on the master machine
- It manages resources and schedule applications running on top of YARN
- It has two components: Scheduler & ApplicationManager
- The Scheduler is responsible for allocating resources to the various running applications
- The ApplicationManager is responsible for accepting job submissions and negotiating the first container for executing the application
- It keeps a track of the heartbeats from the Node Manager

NodeManager

- It is a node level component (one on each node) and runs on each slave machine
- It is responsible for managing containers and monitoring resource utilization in each container
- It also keeps track of node health and log management
- It continuously communicates with ResourceManager to remain up-to-date

2.3.4 Hadoop Ecosystem [32]

So far you would have figured out that Hadoop is neither a programming language nor a service, it is a platform or framework which solves Big Data problems. You can consider

it as a suite which encompasses a number of services for ingesting, storing and analyzing huge data sets along with tools for configuration management.

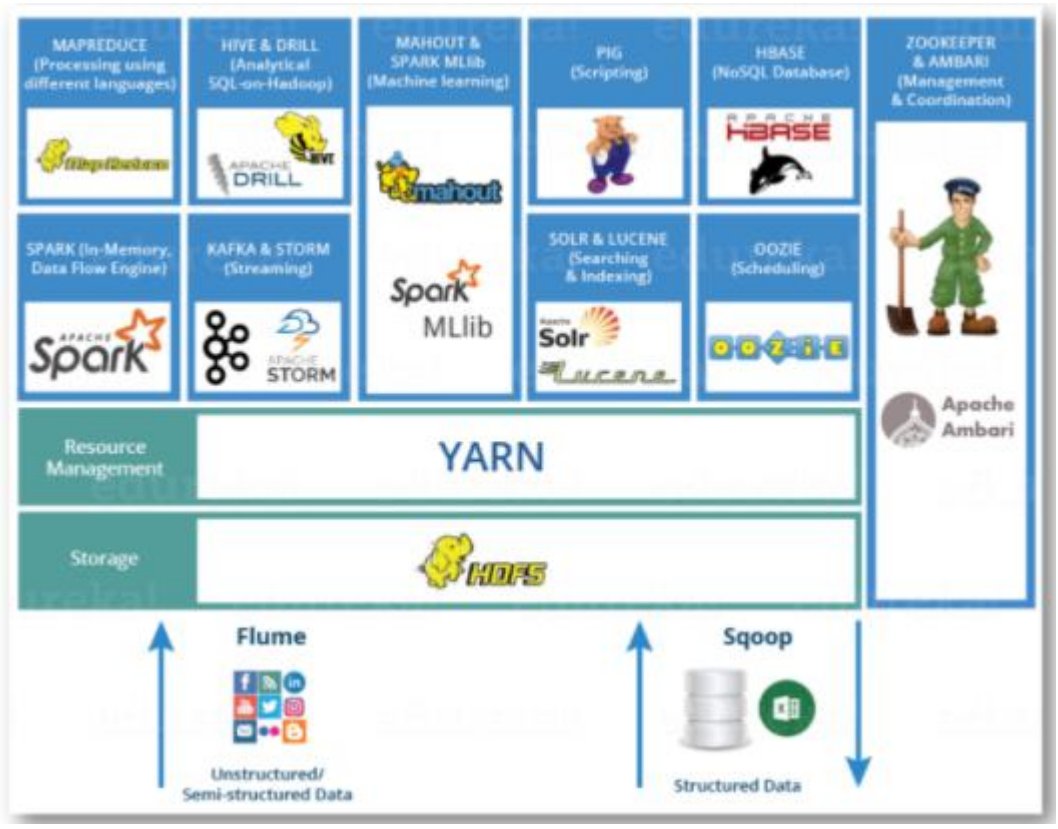


Figure 2.9 Hadoop Ecosystem [32]

2.3.5 Examples of Hadoop - 5 Real-World Use Cases [33]

Here are five examples of Hadoop use cases:

1. Financial services companies use analytics to assess risk, build investment models, and create trading algorithms; Hadoop has been used to help build and run those applications.
2. Retailers use it to help analyze structured and unstructured data to better understand and serve their customers.
3. In the asset-intensive energy industry Hadoop-powered analytics are used for predictive maintenance, with input from Internet of Things (IoT) devices feeding data into big data programs.
4. Telecommunications companies can adapt all the aforementioned use cases. For example, they can use Hadoop-powered analytics to execute predictive maintenance on their infrastructure. Big data analytics can also plan efficient network paths and

recommend optimal locations for new cell towers or other network expansion. To support customer-facing operations telcos can analyze customer behavior and billing statements to inform new service offerings. Examples of Hadoop

5. There are numerous public sector programs, ranging from anticipating and preventing disease outbreaks to crunching numbers to catch tax cheats.

Hadoop is used in these and other big data programs because it is effective, scalable, and is well supported by large vendor and user communities. Hadoop is a de facto standard in big data.

2.3.6 Last.fm Case Study [32]

Last.fm is internet radio and community-driven music discovery service founded in 2002. Users transmit information to Last.fm servers indicating which songs they are listening to. The received data is processed and stored so that, the user can accesses it in the form of charts. Thus, Last.fm can make intelligent taste and compatible decisions for generating recommendations. The data is obtained from one of the two sources stated below:

- **scrobble**: When a user plays a track of his or her own choice and sends the information to Last.fm through a client application.
- **radio listen**: When the user tunes into a Last.fm radio station and streams a song.

Last.fm applications allow users to love, skip or ban each track they listen to. This track listening data is also transmitted to the server.

Last.fm applications allow users to love, skip or ban each track they listen to. This track listening data is also transmitted to the server.

- Over 40M unique visitors and 500M page views each month
- Scrobble stats:
    - Up to 800 scrobbles per second
    - More than 40 million scrobbles per day
    - Over 75 billion scrobbles so far
- Radio stats:
    - Over 10 million streaming hours per month
    - Over 400 thousand unique stations per day
- Each scrobble and radio listen generates at least one log line

Hadoop at Last.FM

- 100 Nodes

- 8 cores per node (dual quad-core)
- 24GB memory per node
- 8TB (4 disks of 2TB each)
- Hive integration to run optimized SQL queries for analysis

Last.FM started using Hadoop in 2006 because of the growth in users from thousands to millions. With the help of Hadoop they processed hundreds of daily, monthly, and weekly jobs including website stats and metrics, chart generation (i.e. track statistics), metadata corrections (e.g. misspellings of artists), indexing for search, combining/formatting data for recommendations, data insights, evaluations & reporting. This helped Last.FM to grow tremendously and figure out the taste of their users, based on which they started recommending music.

2.3.7 Advantages of using Hadoop [34]

Hadoop helps organizations make decisions based on comprehensive analysis of multiple variables and data sets, rather than a small sampling of data or anecdotal incidents. The ability to process large sets of disparate data gives Hadoop users a more comprehensive view of their customers, operations, opportunities, risks, etc. To develop a similar perspective without big data, organizations would need to conduct multiple, limited data analyses then find a way to synthesize the results, which would likely involve a lot of manual effort and subjective analysis.

Here are some other benefits to using Hadoop:

- **Advanced data analysis can be done in-house** – Hadoop makes it practical to work with large data sets and customize the outcome without having to outsource the task to specialist service providers. Keeping operations in-house helps organizations be more agile, while also avoiding the ongoing operational expense of outsourcing. Hadoop Advantages
- **Organizations can fully leverage their data** – One alternative to not using Hadoop is simply not to use all the data and inputs that are available to support business activity. With Hadoop organizations can take full advantage of all their data – structured and unstructured, real-time and historical. Leveraging adds more value to the data itself and improves the return on investment (ROI) for the legacy systems used to collect, process, and store the data, including ERP and CRM systems, social media programs, sensors, industrial automation systems, etc.
- **Run a commodity vs. custom architecture** – Some of the tasks that Hadoop is being used for today were formerly run by MPCC and other specialty, expensive

computer systems. Hadoop commonly runs on commodity hardware. Because it is the de facto big data standard, it is supported by a large and competitive solution provider community, which protects customers from vendor lock-in.

How Hadoop's fundamental problem solving capabilities are applied depends on the use case. As noted, Hadoop data processing is commonly used to support better decision making, provide real-time monitoring for things like machine conditions, threat levels, transaction volumes, etc., and to enable predictive, proactive activity.

It's a thing – Hadoop and the Internet of Things (IoT)

Hadoop can be an important enabling technology for Internet of Things (IoT) projects. In a typical IoT application, a network of sensors or other intelligent devices continually sends current condition data to a platform solution that parses the data, processes what is relevant, and automatically directs an appropriate action (such as shutting down a machine that is at risk of overheating). The sensor and device data is also stored for additional analytics.

IoT programs often produce volumes and types of data that enterprises have never dealt with before. For example, an intelligent factory system can produce millions of sensor readings each day. Organizations that pursue IoT may be quickly pulled into the world of big data. Hadoop can provide a lifeline for efficiently storing, processing, and managing the new data sources.

2.3.8 Disadvantages of using Hadoop [34]

Despite its popularity Hadoop is still an emerging technology, and many of its limitations relate to its newness. The by-products of Hadoop's rapid expansion and evolution include skills gaps, a lack of complementary solutions to support specific needs (e.g., development and debugging tools, native Hadoop support in specific software solutions, etc.). Other criticism stems from Hadoop's status as an open source project, as some professionals consider open source too unstable for business. Other critics say Hadoop is better for storing and aggregating data than it is for processing it.

These disadvantages could be eased as Hadoop becomes more mature. Notably, some of the aforementioned criticisms are really a reflection of the limitations of open-source solutions embedded in the Hadoop ecosystem. For example, the Oozie workflow scheduling utility is often cited as limited and inconvenient to work with, but there are third-party solutions that overcome its limitations and, in some cases, eliminate the need to use Oozie at all.

In addition, Hadoop has some fundamental characteristics that limit its capabilities. Here are some of the most-cited limitations and criticisms regarding Hadoop.

- **Storage requirements** – Hadoop's built-in redundancy duplicates data, thereby requiring more storage resources.
- **Limited SQL support** – Hadoop lacks some of the query functions that SQL database users are accustomed to.
- **Limited native security** – Hadoop does not encrypt data while in storage or when on the network. Further, Hadoop is based on Java, which is a frequent target for malware and other hacks.
- **Component limitations** – There are multiple specific criticisms regarding limitations of Hadoop's four core components (HDFS, YARN, MapReduce and Common). Some of these limitations are overcome by third-party solutions, but the functionality is lacking in Hadoop itself.

2.3.9 Is Hadoop an efficient use of resources? [34]

Once organizations determine that Hadoop will give them a way to work with big data, they often begin to wonder if it is an efficient way. In most cases the answer is yes. Hadoop is also often more cost effective and resource efficient than the methods that are commonly used to maintain enterprise data warehouses (EDWs).

Hadoop is an efficient and cost effective platform for big data because it runs on commodity servers with attached storage, which is a less expensive architecture than a dedicated storage area network (SAN). Commodity Hadoop clusters are also very scalable, which is important because big data programs tend to get bigger as users gain experience and business value from them.

Hadoop not only makes it cost effective to work the big data, but also reduces the costs of maintaining an existing enterprise data warehouse. That's because the essential extract-transport-load (ETL) tasks that are typically performed on the EDW hardware can be offloaded for execution on lower-cost Hadoop clusters. ETL takes a lot of processing cycles, so it is more resource efficient not to execute them on the high-end machines where enterprise data warehouses reside.

The cost and value for using Hadoop depends on the use cases, specific tools and configurations used, and the amount of data in the environment. The supporting tools and solutions used along with core Hadoop technology have a tremendous influence on the

costs to develop and maintain the environment. Deeper insight into the Hadoop ecosystem is provided in the following section.

2.3.10 The business case for Hadoop [34]

The business case for Hadoop depends on the value of information. Hadoop can make more information available, it can analyze more information to support better decision making, and it can make information more valuable by analyzing it in real time. Unlike earlier-generation business intelligence technology, Hadoop helps organizations make sense of both structured and unstructured data. Hadoop can produce new insights because it is able to process new data types, such as social media streams and sensor input. A wide range of organizations have found value in Hadoop by using it to help them better understand their customers, competitors, supply chains, risks and opportunities.

Hadoop can help you take advantage of information from the past, present, and future. Many people think big data is mostly focused on analyzing historical information, which may or may not be relevant in current conditions. While big data solutions often do historical analysis to make predictions and recommendations, it also provides a means to analyze and act on current condition data in real time. For example, Hadoop is the foundation for the recommendation engines some e-commerce retailers use to suggest items while customers are browsing their websites. The recommendations are based on analysis of the specific customer's previous purchase history, what other customers purchased along with the item being viewed, and what's currently trending by performing sentiment analysis on input from social media streams. Those multiple data sources must be processed, analyzed, and converted into actionable information in the short time the customer is on the page. Hadoop makes it all possible, and retailers are reporting sales lifts of between 2 percent and 20 percent as a result. When the sale is made, Hadoop helps financial institutions detect and prevent potential credit card fraud in real time.

Hadoop-driven solutions can also consider current and historical data to guide future activity, such as making assortment planning recommendations for retailers or developing predictive maintenance schedules for production equipment and other assets.

2.3.11 Another way to look at value [34]

The business case for Hadoop is different for every business. To begin to understand whether Hadoop is a fit for your organization and how it could provide value, ask:

- What is the value of better decision making?

- Do past customer behaviors, business conditions, risk factors, etc. have any bearing on current or future performance?
- Would faster decision-making help the business?
- Would predictive maintenance and reducing unplanned downtime be valuable?
- What incremental value would improve demand forecasting provide?
- Could we benefit from sentiment analysis?
- Would real-time monitoring assist information security or compliance requirements?
- Are we getting all we can out of the structured and unstructured data we have?

Hadoop's use cases and benefits are very broad. In many cases, Hadoop won't be introduced as a replacement technology, but instead will be used to do things that haven't been done before. Therefore it is helpful to examine Hadoop's benefits, limitations, and alternatives from a business, not technical, perspective.

2.3.12 What does Hadoop replace? [34]

Hadoop is commonly used to support better decision making. Therefore it often complements the data processing and reporting systems that are already used, rather than replacing them. For example, Tableau is a popular business intelligence tool that organizations use to process and visualize a variety of data. Tableau supports Hadoop and provides another option to output Hadoop-driven data analysis. Excel can also process data imported from Hadoop, however, more sophisticated, big data-oriented solutions support more capabilities. The Hadoop ecosystem is continually expanding with new solutions to help users take advantage of Hadoop in new ways.

Hadoop does replace the need to have clusters of customized, high-performance computers that are supported by large teams of technicians and data scientists to turn data into actionable information. Hadoop is an alternative to using massively parallel processing (MPP) database structures, which tend to be custom built and expensive. Hadoop can also replace traditional silo-based IT architectures, or at least provide a way for the silos to interact and serve as a single source for data.

2.3.13 Problems that Hadoop solves [34]

Hadoop solves the fundamental problem of processing large sets of structured and unstructured data that come from disparate sources and reside in different systems. More specifically, Hadoop addresses the challenges of scale, latency, and comprehensiveness when dealing with large data sets.

Scale – Hadoop makes it practical to process large sets of data on commodity computers. It also solves the problem of getting an accurate, single view of structured and unstructured data that is held in multiple systems.
Latency – Hadoop can process large data sets much faster than traditional methods, so organizations can act on the data while it is still current. For example, recommendation engines suggest complementary items or special offers in real time. In the past, organizations might have run a report on what customers purchased, analyzed the results, then sent a follow-up email days or weeks later suggesting the same complementary items. There is also tremendous value to removing latency when monitoring for network intrusion, financial fraud and other threats.
Comprehensiveness - One of the things that set Hadoop apart is its ability to process different types of data. Besides traditional, structured data, sometimes referred to as data at rest, Hadoop can sort and process data in motion, such as input from sensors, location data, social media channels, and metadata from video and customer contact systems. It can also perform clickstream data analysis. The comprehensiveness creates more visibility and a deeper understanding of what is being studied.

These Hadoop characteristics have also been expressed as the three Vs – volume, velocity, and variety.

2.4 HADOOP INSTALLATION [35]

Environment required for Hadoop: The production environment of Hadoop is UNIX, but it can also be used in Windows using Cygwin. Java 1.6 or above is needed to run Map Reduce Programs. For Hadoop installation from tar ball on the UNIX environment you need

1. Java Installation
2. SSH installation
3. Hadoop Installation and File Configuration

2.4.1 Java Installation [35]

Step 1. Type "java -version" in prompt to find if the java is installed or not. If not then download java from http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html . The tar filejdk-7u71-linux-x64.tar.gz will be downloaded to your system.

Step 2. Extract the file using the below command

```
#tar zxf jdk-7u71-linux-x64.tar.gz
```

Step 3. To make java available for all the users of UNIX move the file to /usr/local and set the path. In the prompt switch to root user and then type the command below to move the jdk to /usr/lib.

```
# mv jdk1.7.0_71 /usr/lib/
```

Now in ~/.bashrc file add the following commands to set up the path.

```
# export JAVA_HOME=/usr/lib/jdk1.7.0_71
# export PATH=PATH:$JAVA_HOME/bin
```

Now, you can check the installation by typing "java -version" in the prompt.

2.4.2 SSH Installation [35]

SSH is used to interact with the master and slaves computer without any prompt for password. First of all create a Hadoop user on the master and slave systems

```
# useradd hadoop
# passwd Hadoop
```

To map the nodes open the hosts file present in /etc/ folder on all the machines and put the ip address along with their host name.

```
# vi /etc/hosts
```

Enter the lines below

```
190.12.1.114   hadoop-master
190.12.1.121   hadoop-salve-one
190.12.1.143   hadoop-slave-two
```

Set up SSH key in every node so that they can communicate among themselves without password. Commands for the same are:

```
# su hadoop
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
$ chmod 0600 ~/.ssh/authorized_keys
$ exit
```

2.4.3 Hadoop Installation [35]

Hadoop can be downloaded from http://developer.yahoo.com/hadoop/tutorial/module3.html

Now extract the Hadoop and copy it to a location.

```
$ mkdir /usr/hadoop
$ sudo tar vxzf  hadoop-2.2.0.tar.gz ?c /usr/hadoop
```

Change the ownership of Hadoop folder

```
$sudo chown -R hadoop  usr/hadoop
```

Change the Hadoop configuration files:

All the files are present in /usr/local/Hadoop/etc/hadoop

1) In hadoop-env.sh file add

```
export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
```

2) In core-site.xml add following between configuration tabs,

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://hadoop-master:9000</value>
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
</configuration>
```

3) In hdfs-site.xmladd following between configuration tabs,

```
<configuration>
<property>
<name>dfs.data.dir</name>
<value>usr/hadoop/dfs/name/data</value>
<final>true</final>
</property>
<property>
<name>dfs.name.dir</name>
<value>usr/hadoop/dfs/name</value>
<final>true</final>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

4) Open the Mapred-site.xml and make the change as shown below

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>hadoop-master:9001</value>
</property>
</configuration>
```

5) Finally, update your $HOME/.bahsrc

```
cd $HOME
vi .bashrc
Append following lines in the end and save and exit
#Hadoop variables
export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
export HADOOP_INSTALL=/usr/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
```

On the slave machine install Hadoop using the command below

```
# su hadoop
$ cd /opt/hadoop
$ scp -r hadoop hadoop-slave-one:/usr/hadoop
$ scp -r hadoop hadoop-slave-two:/usr/Hadoop
```

Configure master node and slave node

```
$ vi etc/hadoop/masters
hadoop-master

$ vi etc/hadoop/slaves
hadoop-slave-one
hadoop-slave-two
```

After this format the name node and start all the deamons

```
# su hadoop
$ cd /usr/hadoop
$ bin/hadoop namenode -format

$ cd $HADOOP_HOME/sbin
$ start-all.sh
```

The easiest step is the usage of cloudera as it comes with all the stuffs pre-installed which can be downloaded from http://content.udacity-data.com/courses/ud617/Cloudera-Udacity-Training-VM-4.1.1.c.zip

2.5 HADOOP MODULES [35]

2.5.1 HDFS [35]

What is HDFS

Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.

It is cost effective as it uses commodity hardware. It involves the concept of blocks, data nodes and node name.

Where to use HDFS

- **Very Large Files**: Files should be of hundreds of megabytes, gigabytes or more.

- **Streaming Data Access**: The time to read whole data set is more important than latency in reading the first. HDFS is built on write-once and read-many-times pattern.
- **Commodity Hardware**: It works on low cost hardware.

Where not to use HDFS

- **Low Latency data access**: Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.
- **Lots Of Small Files**: The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.
- **Multiple Writes**: It should not be used when we have to write multiple times.

HDFS Concepts

1. **Blocks**: A Block is the minimum amount of data that it can read or write.HDFS blocks are 128 MB by default and this is configurable.Files n HDFS are broken into block-sized chunks,which are stored as independent units.Unlike a file system, if the file is in HDFS is smaller than block size, then it does not occupy full block?s size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only.The HDFS block size is large just to minimize the cost of seek.

2. **Name Node**: HDFS works in master-worker pattern where the name node acts as master.Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block.The metadata are small, so it is stored in the memory of name node,allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently,so all this information is handled bya single machine. The file system operations like opening, closing, renaming etc. are executed by it.

3. **Data Node**: They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

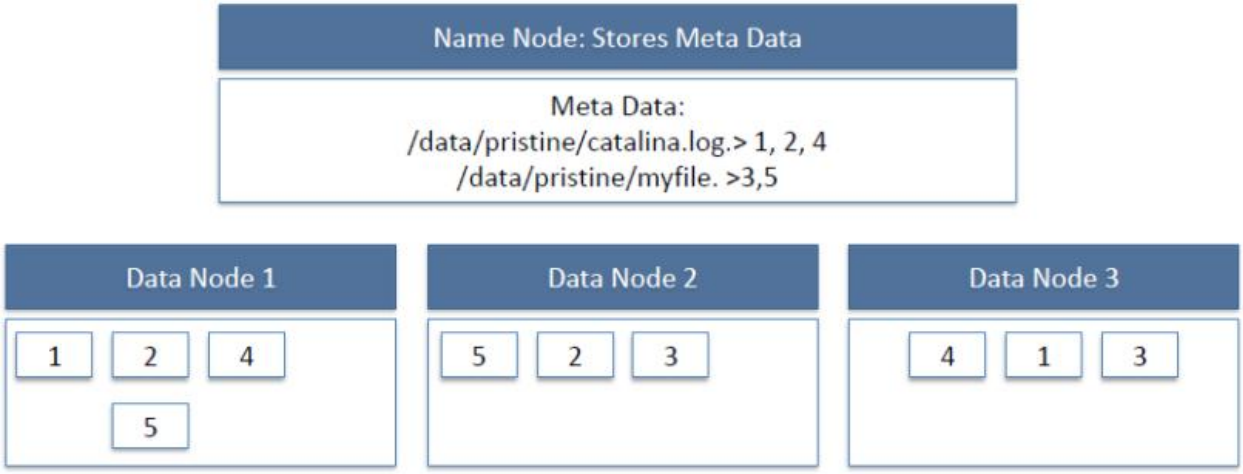


Figure 2.10 HDFS DataNode and NameNode Image [35]

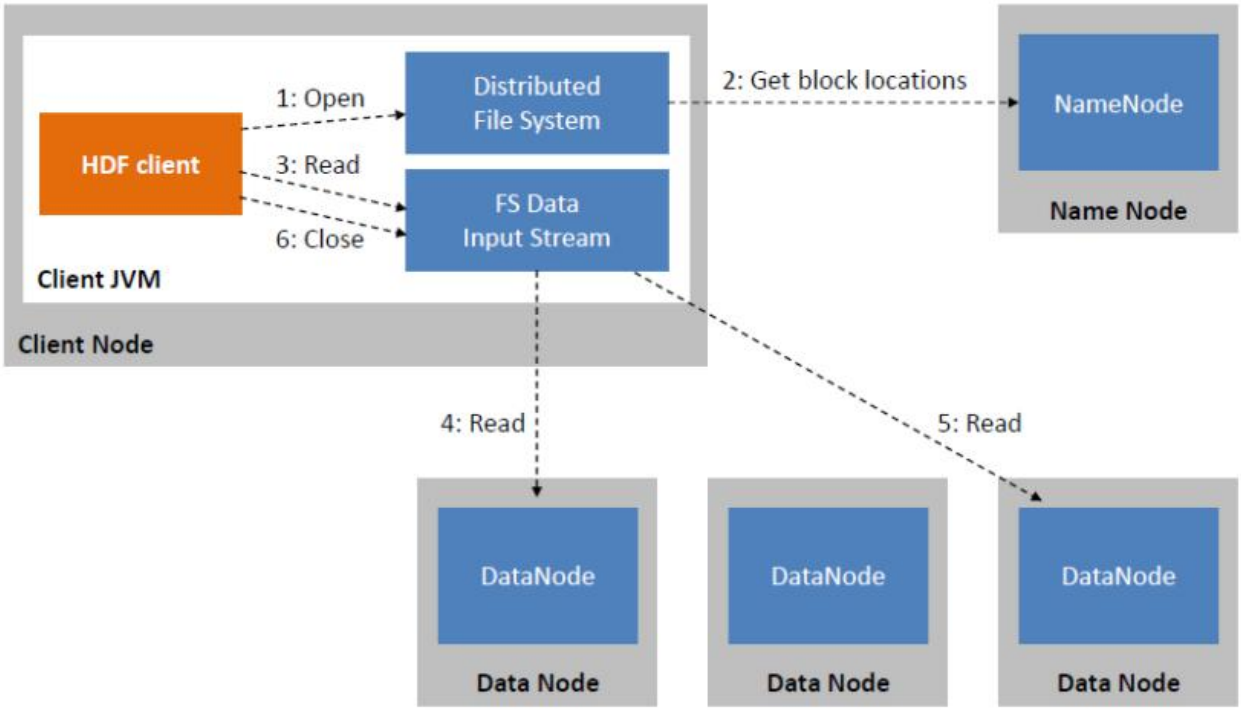


Figure 2.11 HDFS Read Image [35]

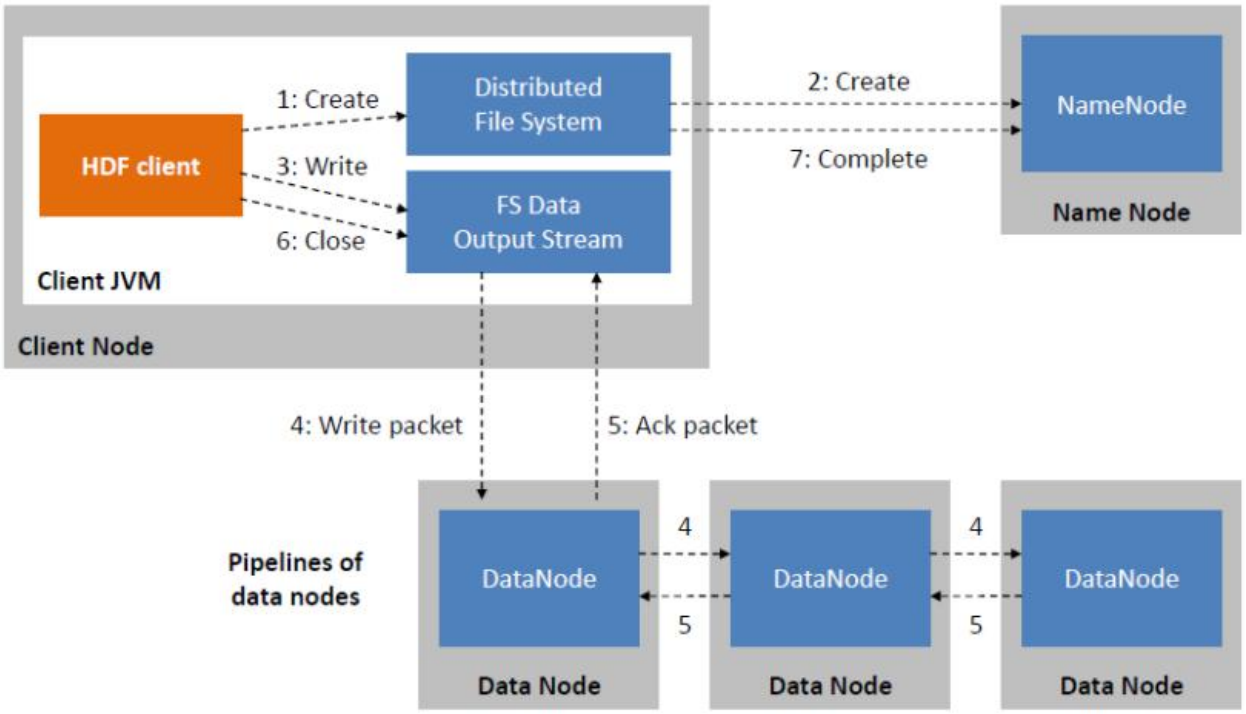


Figure 2.12 HDFS Write Image [35]

Since all the metadata is stored in name node, it is very important. If it fails the file system cannot be used as there would be no way of knowing how to reconstruct the files from blocks present in data node. To overcome this, the concept of secondary name node arises.

Secondary Name Node: It is a separate physical machine which acts as a helper of name node. It performs periodic check points. It communicates with the name node and take snapshot of meta data which helps minimize downtime and loss of data.

Starting HDFS

The HDFS should be formatted initially and then started in the distributed mode. Commands are given below.

To Format **$ hadoop namenode -format**

To Start **$ start-dfs.sh**

HDFS Basic File Operations

1. Putting data to HDFS from local file system
   - First create a folder in HDFS where data can be put form local file system.
     $ hadoop fs -mkdir /user/test

   - Copy the file "data.txt" from a file kept in local folder /usr/home/Desktop to HDFS folder /user/ test
     $ hadoop fs -copyFromLocal /usr/home/Desktop/data.txt /user/test

   - Display the content of HDFS folder
     $ Hadoop fs -ls /user/test

2. Copying data from HDFS to local file system
   - $ hadoop fs -copyToLocal /user/test/data.txt /usr/bin/data_copy.txt

3. Compare the files and see that both are same
   - $ md5 /usr/bin/data_copy.txt /usr/home/Desktop/data.txt

Recursive deleting

- hadoop fs -rmr <arg>

Example:

- hadoop fs -rmr /user/sonoo/

HDFS Other commands

The below is used in the commands

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system

- put <localSrc><dest>
  Copies the file or directory from the local file system identified by localSrc to dest within the DFS.

- copyFromLocal <localSrc><dest>
  Identical to -put

- copyFromLocal <localSrc><dest>
  Identical to -put

- moveFromLocal <localSrc><dest>
  Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.

- get [-crc] <src><localDest>
  Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.

- cat <filen-ame>
  Displays the contents of filename on stdout.

- moveToLocal <src><localDest>
  Works like -get, but deletes the HDFS copy on success.

- setrep [-R] [-w] rep <path>
  Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)

- touchz <path>
  Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.

- test -[ezd] <path>
  Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

- stat [format] <path>
  Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).

2.5.2 YARN [35]

What is YARN

Yet Another Resource Manager takes programming to the next level beyond Java , and makes it interactive to let another application Hbase, Spark etc. to work on it.Different Yarn applications can co-exist on the same cluster so MapReduce, Hbase, Spark all can run at the same time bringing great benefits for manageability and cluster utilization.

Components of YARN

- **Client**: For submitting MapReduce jobs.
- **Resource Manager**: To manage the use of resources across the cluster
- **Node Manager**: For launching and monitoring the computer containers on machines in the cluster.
- **Map Reduce Application Master**: Checks tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager, and managed by the node managers.

Jobtracker & Tasktrackerwere were used in previous version of Hadoop, which were responsible for handling resources and checking progress management. However, Hadoop 2.0 has Resource manager and NodeManager to overcome the shortfall of Jobtracker & Tasktracker.

Benefits of YARN

- **Scalability**: Map Reduce 1 hits ascalability bottleneck at 4000 nodes and 40000 task, but Yarn is designed for 10,000 nodes and 1 lakh tasks.
- **Utiliazation**: Node Manager manages a pool of resources, rather than a fixed number of the designated slots thus increasing the utilization.
- **Multitenancy**: Different version of MapReduce can run on YARN, which makes the process of upgrading MapReduce more manageable.

2.5.3 MapReduce [35]

To take the advantage of parallel processing of Hadoop, the query must be in MapReduce form. The MapReduce is a paradigm which has two phases, the mapper phase and the

reducer phase. In the Mapper the input is given in the form of key value pair. The output of the mapper is fed to the reducer as input. The reducer runs only after the mapper is over. The reducer too takes input in key value format and the output of reducer is final output.

Steps in Map Reduce

- Map takes a data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- Output of sort and shuffle will be sent to reducer phase. Reducer will perform a defined function on list of values for unique keys and Final output will<key, value> will be stored/displayed.

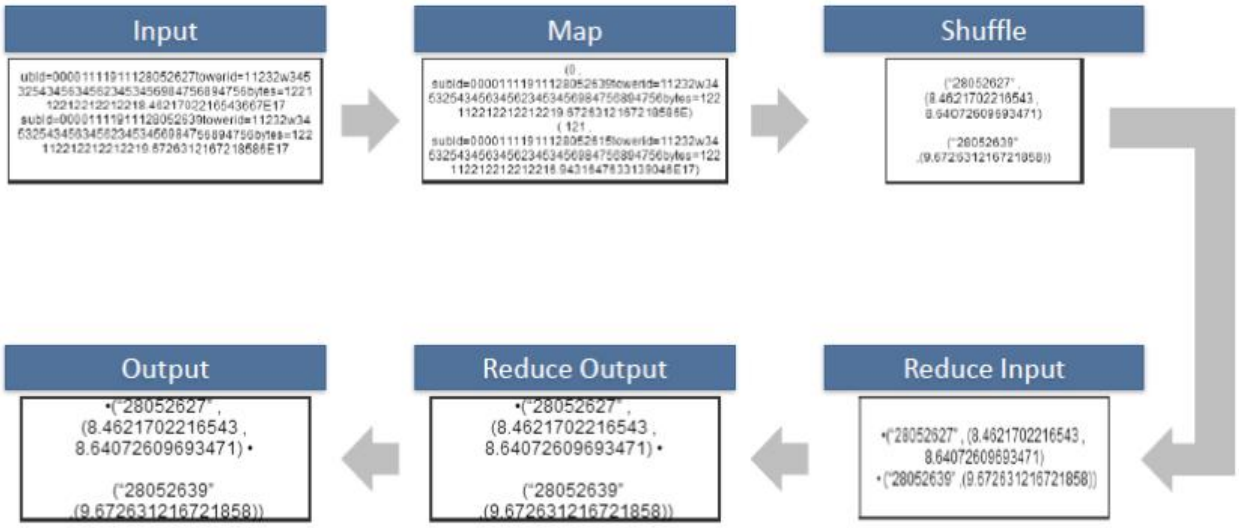


Figure 2.13 Steps in Map Reduce [35]

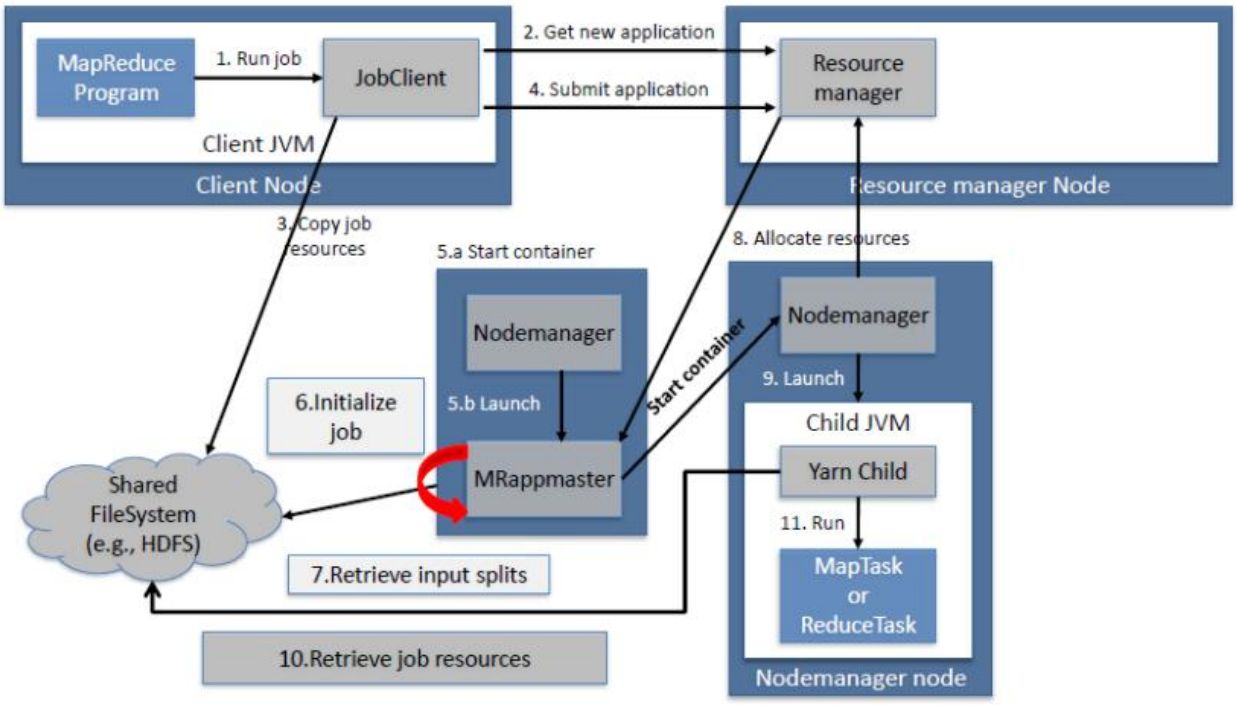


Figure 2.14 Map Reduce Architecture [35]

How Many Maps

The size of data to be processed decides the number of maps required. For example, we have 1000 MB data and block size is 64 MB then we need 16 mappers.

Sort and Shuffle

The sort and shuffle occur on the output of mapper and before the reducer. When the mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each mapper <k2,v2> , we collect all the values for each unique key k2. This output from the shuffle phase in the form of <k2,list(v2)> is sent as input to reducer phase.

MapReduce Example

Use Case

Find the number of occurrences of the word using Map Reduce in a text file

Solution:

Step 1: Upload the file on HDFS data.txt from /usr/Desktop(local path) to /Hadoop/data (Hadoop folder).

$hadoop fs ?put /usr/Desktop/data.txt /Hadoop/data

Step 2: Write the Map reduce program using eclipse and make the jar of it and name it count.

File: wc_mapper.java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class wc_mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
         Reporter reporter) throws IOException{
      String line = value.toString();
      StringTokenizer  tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()){
         word.set(tokenizer.nextToken());
         output.collect(word, one);
      }
   }

}
```

File: wc_reducer.java

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class wc_reducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
 Reporter reporter) throws IOException {
int sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
output.collect(key,new IntWritable(sum));
}
}
```

File: wc_runner.java

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class wc_runner {
   public static void main(String[] args) throws IOException{
      JobConf conf = new JobConf(wc_runner.class);
      conf.setJobName("WordCount");
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      conf.setMapperClass(wc_mapper.class);
      conf.setCombinerClass(wc_reducer.class);
      conf.setReducerClass(wc_reducer.class);
      conf.setInputFormat(TextInputFormat.class);
      conf.setOutputFormat(TextOutputFormat.class);
      FileInputFormat.setInputPaths(conf,new Path(args[0]));
      FileOutputFormat.setOutputPath(conf,new Path(args[1]));
      JobClient.runJob(conf);
   }
}
```

Step 3: Run the jar file

$hadoop jar count.jar WordCount /Hadoop/data.txt/user/root/example_count

The output is stored in example_countfolder.

CHAPTER 3

BIG DATA STORAGE AND ANALYTICS

3.1 BIG DATA STORAGE CONCEPTS [26]

Data acquired from external sources is often not in a format or structure that can be directly processed. To overcome these incompatibilities and prepare data for storage and processing, data wrangling is necessary. Data wrangling includes steps to filter, cleanse and otherwise prepare the data for downstream analysis. From a storage perspective, a copy of the data is first stored in its acquired format, and, after wrangling, the prepared data needs to be stored again. Typically, storage is required whenever the following occurs:

• external datasets are acquired, or internal data will be used in a Big Data environment
• data is manipulated to be made amenable for data analysis
• data is processed via an ETL activity, or output is generated as a result of an analytical operation

Due to the need to store Big Data datasets, often in multiple copies, innovative storage strategies and technologies have been created to achieve cost-effective and highly scalable storage solutions. In order to understand the underlying mechanisms behind Big Data storage technology, the following topics are introduced in this chapter:
• clusters
• file systems and distributed files systems
• NoSQL
• sharding
• replication
• CAP theorem
• ACID
• BASE

3.1.1 Clusters [26]

In computing, a cluster is a tightly coupled collection of servers, or nodes. These servers usually have the same hardware specifications and are connected together via a network to work as a single unit, as shown in Figure 3.1. Each node in the cluster has its own dedicated resources, such as memory, a processor, and a hard drive. A cluster can execute

a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster.

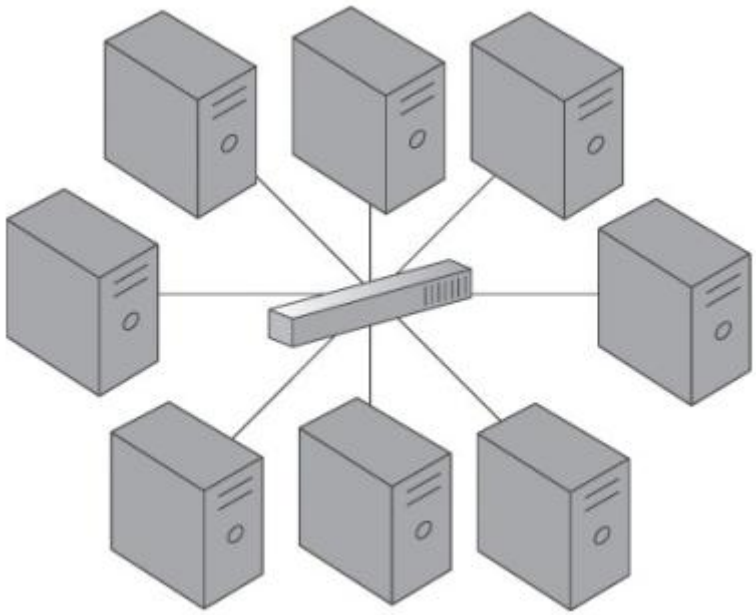


Figure 3.1 The symbol used to represent a cluster [26]

3.1.2 File Systems and Distributed File Systems [26]

A file system is the method of storing and organizing data on a storage device, such as flash drives, DVDs and hard drives. A file is an atomic unit of storage used by the file system to store data. A file system provides a logical view of the data stored on the storage device and presents it as a tree structure of directories and files as pictured in Figure 3.2. Operating systems employ file systems to store and retrieve data on behalf of applications. Each operating system provides support for one or more file systems, for example NTFS on Microsoft Windows and ext on Linux.

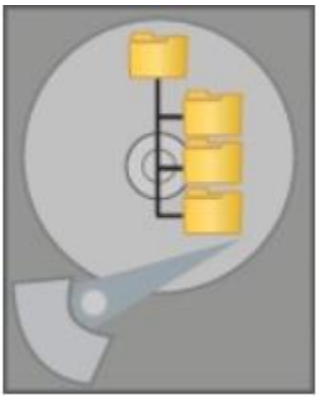


Figure 3.2 The symbol used to represent a file system [26]

A distributed file system is a file system that can store large files spread across the nodes of a cluster, as illustrated in Figure 3.3. To the client, files appear to be local; however, this is only a logical view as physically the files are distributed throughout the cluster. This local view is presented via the distributed file system and it enables the files to be accessed from multiple locations. Examples include the Google File System (GFS) and Hadoop Distributed File System (HDFS).

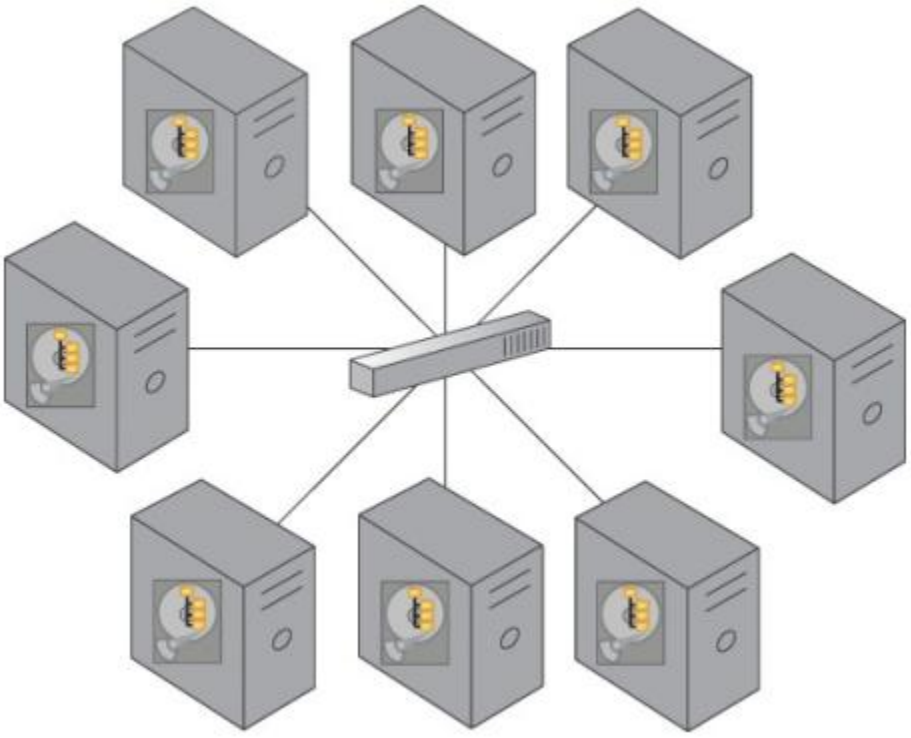


Figure 3.3 The symbol used to represent distributed file systems [26]

3.1.3 NoSQL [26]

A Not-only SQL (NoSQL) database is a non-relational database that is highly scalable, fault-tolerant and specifically designed to house semi-structured and unstructured data. A NoSQL database often provides an API-based query interface that can be called from within an application. NoSQL databases also support query languages other than Structured Query Language (SQL) because SQL was designed to query structured data stored within a relational database. As an example, a NoSQL database that is optimized to store XML files will often use XQuery as the query language. Likewise, a NoSQL database designed to store RDF data will use SPARQL to query the relationships it contains. That being said, there are some NoSQL databases that also provide an SQL-like query interface, as shown in Figure 3.4.

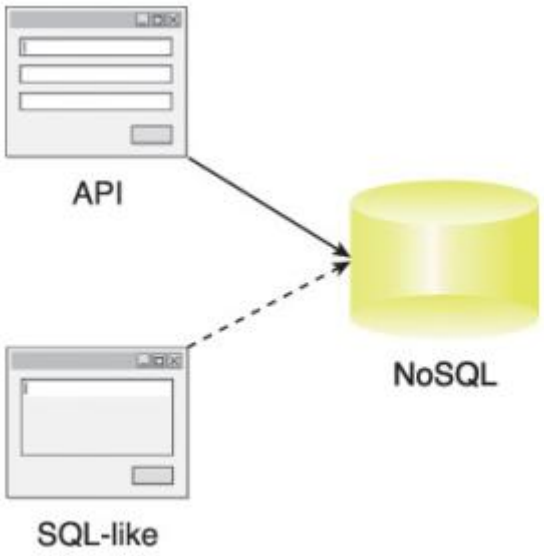


Figure 3.4 A NoSQL database can provide an API- or SQL-like query interface [26]

3.1.4 Sharding [26]

Sharding is the process of horizontally partitioning a large dataset into a collection of smaller, more manageable datasets called shards. The shards are distributed across multiple nodes, where a node is a server or a machine (Figure 3.5). Each shard is stored on a separate node and each node is responsible for only the data stored on it. Each shard shares the same schema, and all shards collectively represent the complete dataset.

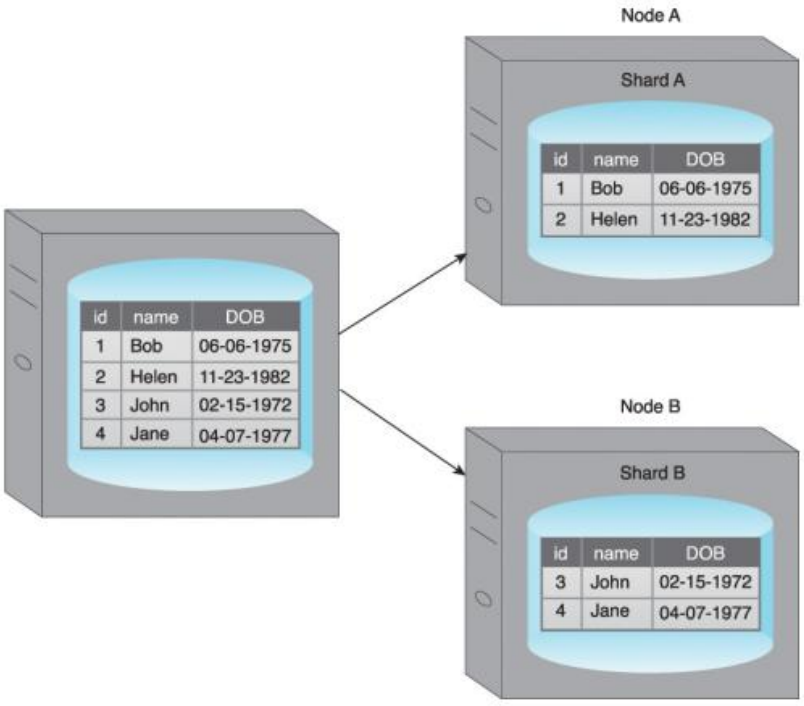


Figure 3.5 An example of sharding where a dataset is spread across Node A and Node B, resulting in Shard A and Shard B, respectively. [26]

Sharding is often transparent to the client, but this is not a requirement. Sharding allows the distribution of processing loads across multiple nodes to achieve horizontal scalability. Horizontal scaling is a method for increasing a system's capacity by adding similar or higher capacity resources alongside existing resources. Since each node is responsible for only a part of the whole dataset, read/write times are greatly improved.

Figure 3.6 presents an illustration of how sharding works in practice:

1. Each shard can independently service reads and writes for the specific subset of data that it is responsible for.
2. Depending on the query, data may need to be fetched from both shards.

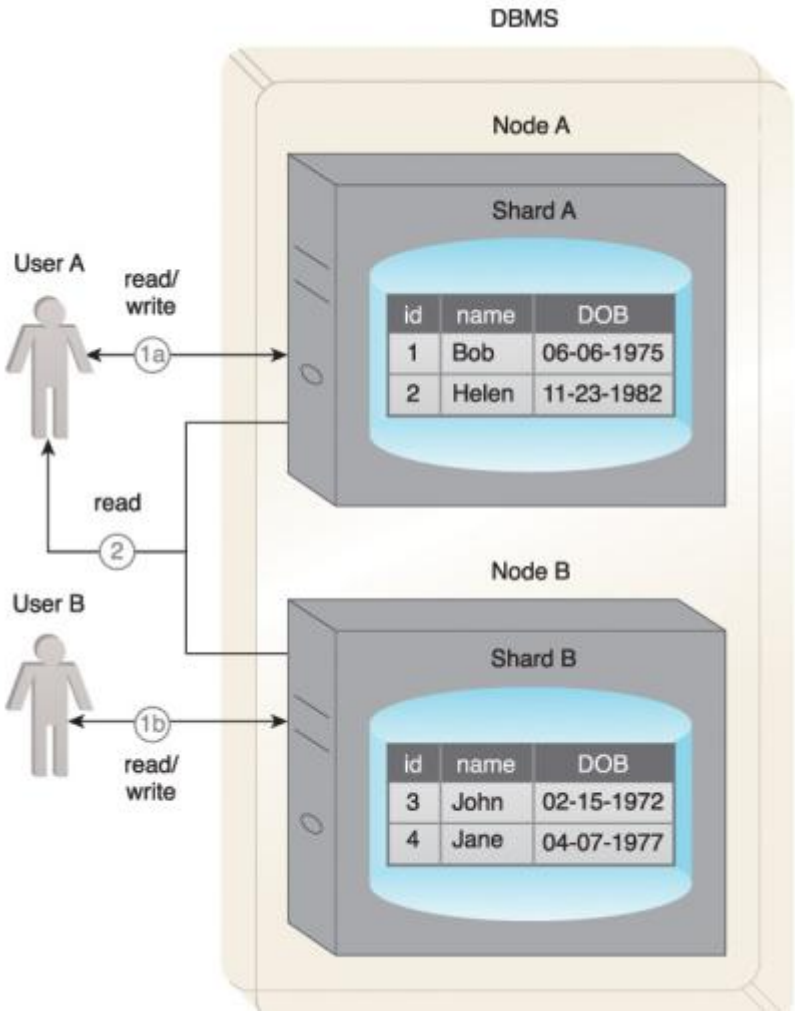


Figure 3.6 A sharding example where data is fetched from both Node A and Node B [26]

A benefit of sharding is that it provides partial tolerance toward failures. In case of a node failure, only data stored on that node is affected. With regards to data partitioning, query patterns need to be taken into account so that shards themselves do not become performance bottlenecks. For example, queries requiring data from multiple shards will impose performance penalties. Data locality keeps commonly accessed data co-located on a single shard and helps counter such performance issues.

3.1.5 Replication [26]

Replication stores multiple copies of a dataset, known as replicas, on multiple nodes (Figure 3.7). Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is also achieved since data redundancy ensures that data is not lost when an individual node fails. There are two different methods that are used to implement replication:

• master-slave
• peer-to-peer

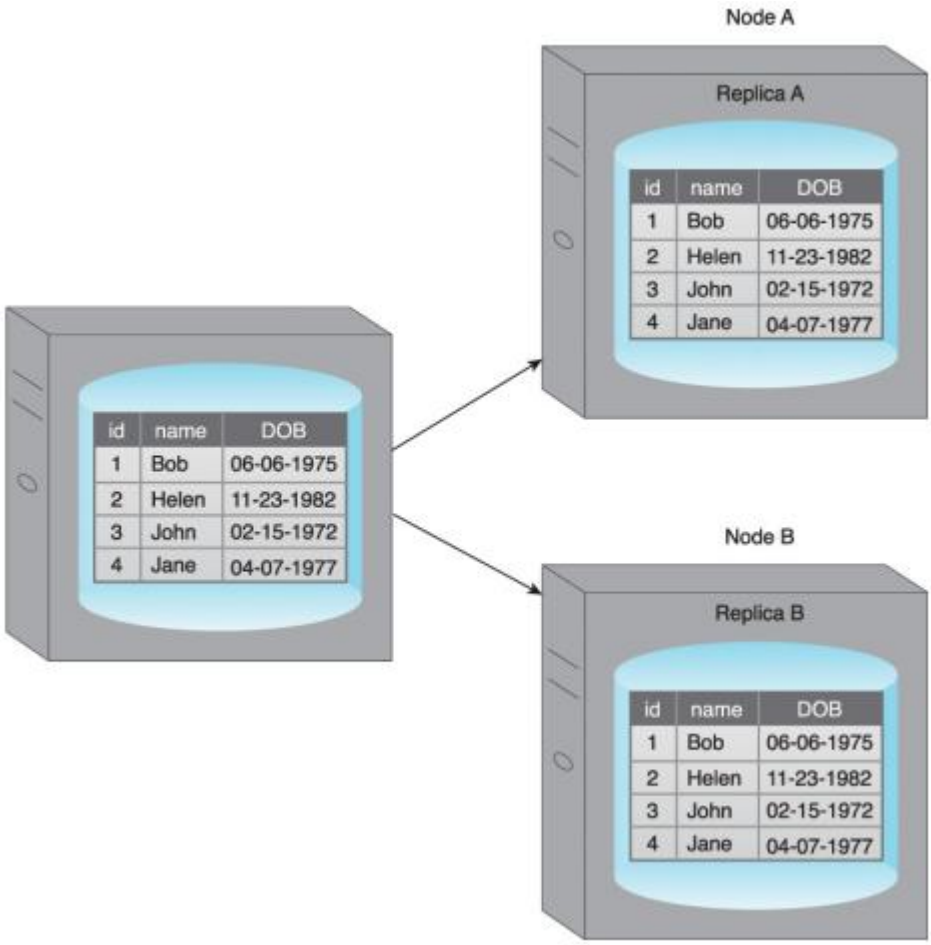


Figure 3.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B [26]

Master-Slave

During master-slave replication, nodes are arranged in a master-slave configuration, and all data is written to a master node. Once saved, the data is replicated over to multiple slave nodes. All external write requests, including insert, update and delete, occur on the master node, whereas read requests can be fulfilled by any slave node. In Figure 3.8, writes are managed by the master node and data can be read from either Slave A or Slave B.

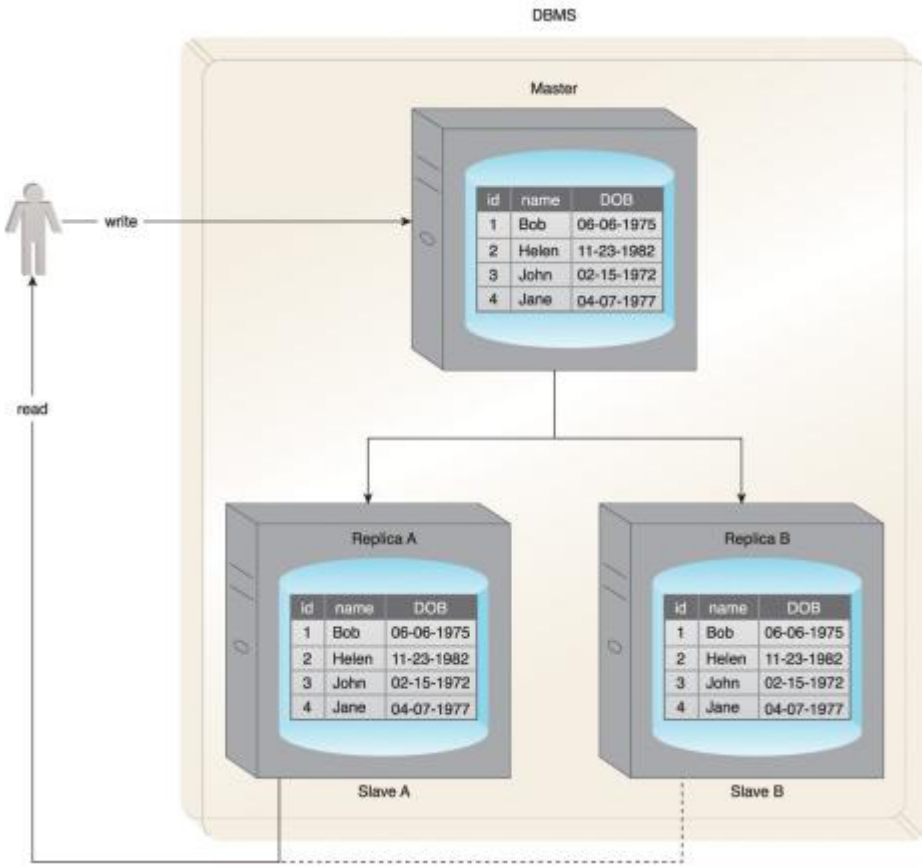


Figure 3.8 An example of master-slave replication where Master A is the single point of contact for all writes, and data can be read from Slave A and Slave B [26]

Master-slave replication is ideal for read intensive loads rather than write intensive loads since growing read demands can be managed by horizontal scaling to add more slave nodes. Writes are consistent, as all writes are coordinated by the master node. The implication is that write performance will suffer as the amount of writes increases. If the master node fails, reads are still possible via any of the slave nodes.

A slave node can be configured as a backup node for the master node. In the event that the master node fails, writes are not supported until a master node is reestablished. The master node is either resurrected from a backup of the master node, or a new master node is chosen from the slave nodes.

One concern with master-slave replication is read inconsistency, which can be an issue if a slave node is read prior to an update to the master being copied to it. To ensure read consistency, a voting system can be implemented where a read is declared consistent if the majority of the slaves contain the same version of the record. Implementation of such a voting system requires a reliable and fast communication mechanism between the slaves.

Figure 3.9 illustrates a scenario where read inconsistency occurs.

1. User A updates data.

2. The data is copied over to Slave A by the Master.
3. Before the data is copied over to Slave B, User B tries to read the data from Slave B, which results in an inconsistent read.
4. The data will eventually become consistent when Slave B is updated by the Master.

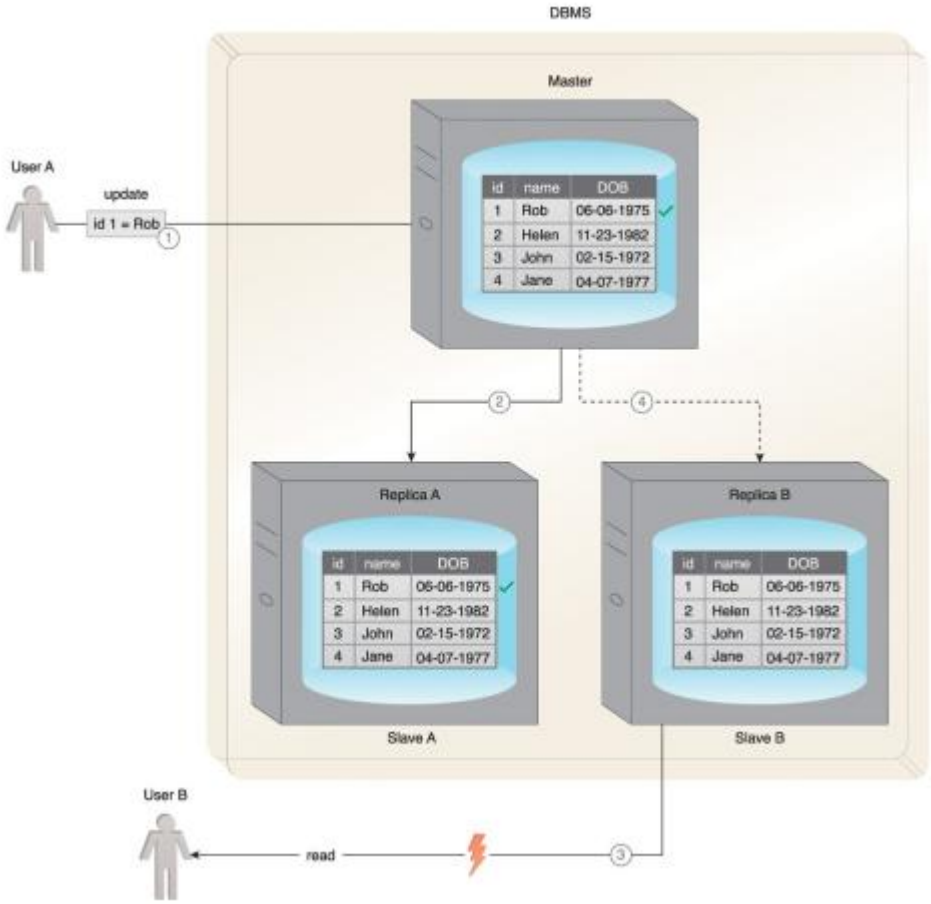


Figure 3.9 An example of master-slave replication where read inconsistency occurs. [26]

Peer-to-Peer

With peer-to-peer replication, all nodes operate at the same level. In other words, there is not a master-slave relationship between the nodes. Each node, known as a peer, is equally capable of handling reads and writes. Each write is copied to all peers, as illustrated in Figure 3.10.

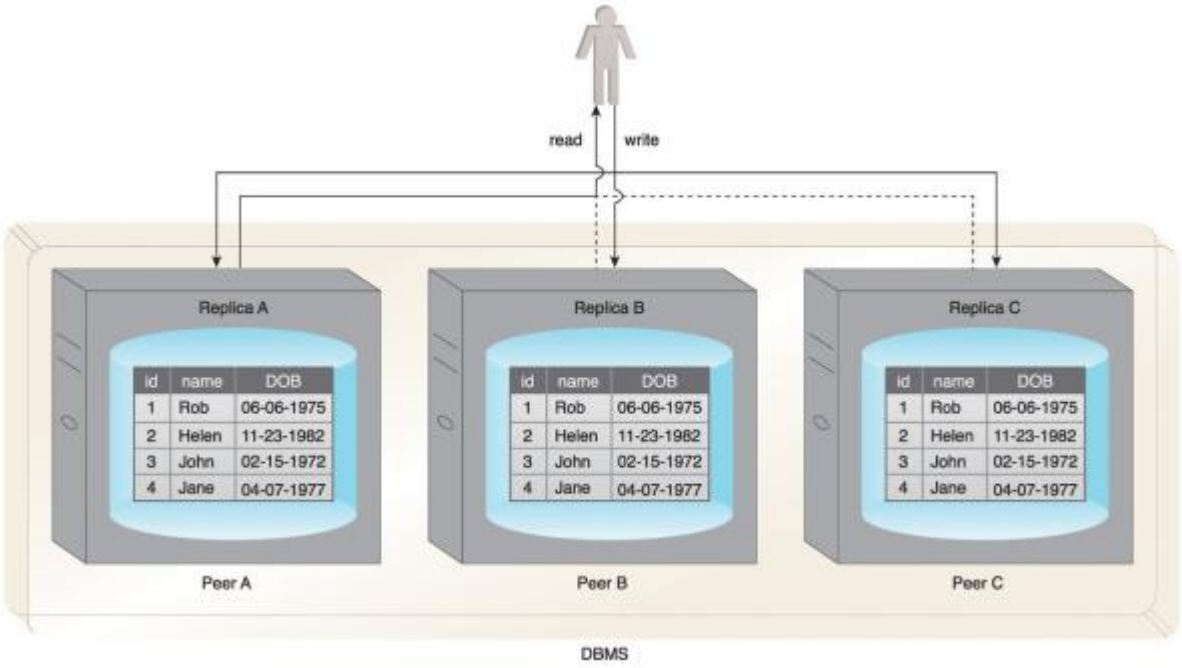


Figure 3.10 Writes are copied to Peers A, B and C simultaneously. Data is read from Peer A, but it can also be read from Peers B or C. [26]

Peer-to-peer replication is prone to write inconsistencies that occur as a result of a simultaneous update of the same data across multiple peers. This can be addressed by implementing either a pessimistic or optimistic concurrency strategy.

- Pessimistic concurrency is a proactive strategy that prevents inconsistency. It uses locking to ensure that only one update to a record can occur at a time. However, this is detrimental to availability since the database record being updated remains unavailable until all locks are released.
- Optimistic concurrency is a reactive strategy that does not use locking. Instead, it allows inconsistency to occur with knowledge that eventually consistency will be achieved after all updates have propagated.

With optimistic concurrency, peers may remain inconsistent for some period of time before attaining consistency. However, the database remains available as no locking is involved. Like master-slave replication, reads can be inconsistent during the time period when some of the peers have completed their updates while others perform their updates. However, reads eventually become consistent when the updates have been executed on all peers.

To ensure read consistency, a voting system can be implemented where a read is declared consistent if the majority of the peers contain the same version of the record. As

previously indicated, implementation of such a voting system requires a reliable and fast communication mechanism between the peers.

Figure 3.11 demonstrates a scenario where an inconsistent read occurs.

1. User A updates data.
2. a. The data is copied over to Peer A.
   b. The data is copied over to Peer B.
3. Before the data is copied over to Peer C, User B tries to read the data from Peer C, resulting in an inconsistent read.
4. The data will eventually be updated on Peer C, and the database will once again become consistent.

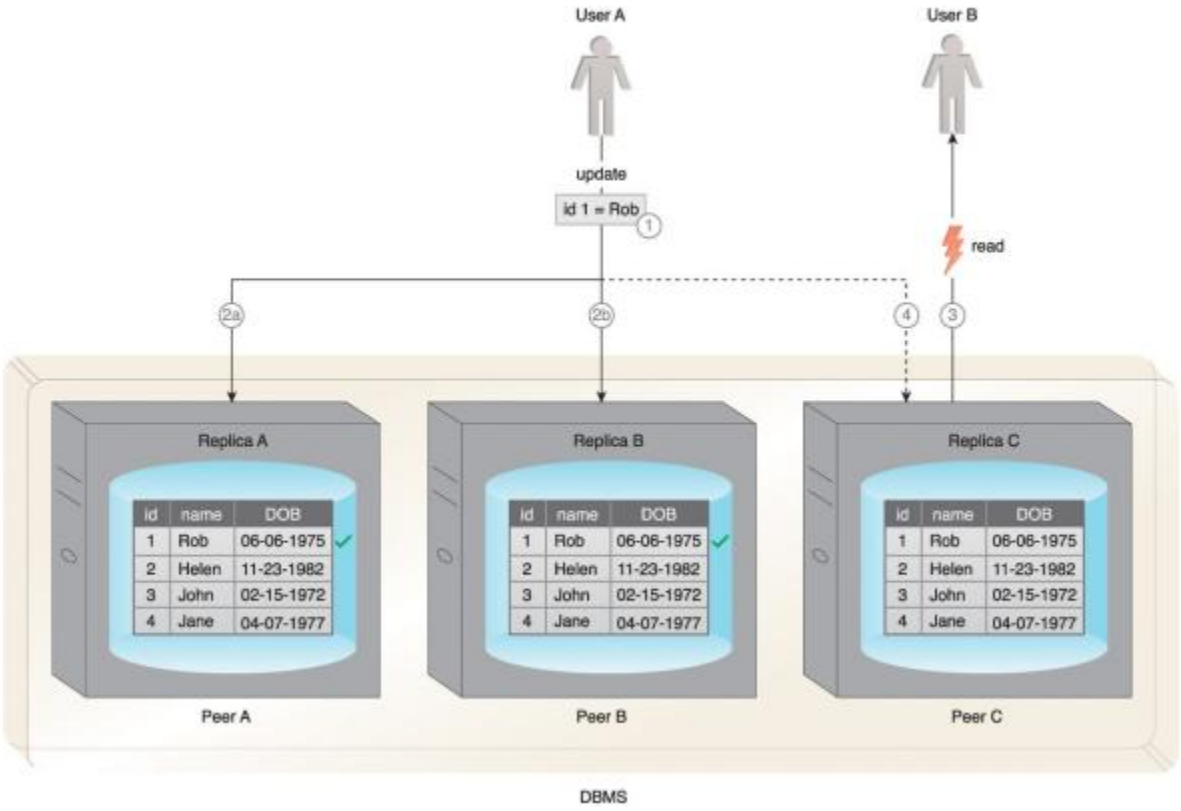


Figure 3.11 An example of peer-to-peer replication where an inconsistent read occurs. [26]

3.1.6 Sharding and Replication [26]

To improve on the limited fault tolerance offered by sharding, while additionally benefiting from the increased availability and scalability of replication, both sharding and replication can be combined, as shown in Figure 3.12.

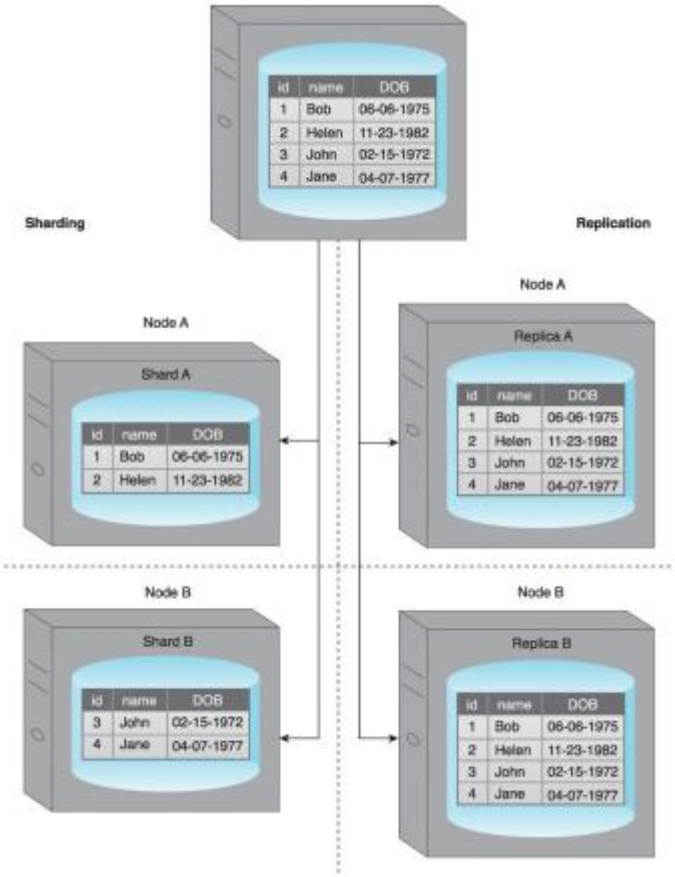


Figure 3.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches. [26]

This section covers the following combinations:

- sharding and master-slave replication
- sharding and peer-to-peer replication

Combining Sharding and Master-Slave Replication

When sharding is combined with master-slave replication, multiple shards become slaves of a single master, and the master itself is a shard. Although this results in multiple masters, a single slave-shard can only be managed by a single master-shard.

Write consistency is maintained by the master-shard. However, if the master-shard becomes non-operational or a network outage occurs, fault tolerance with regards to write operations is impacted. Replicas of shards are kept on multiple slave nodes to provide scalability and fault tolerance for read operations.

In Figure 3.13:

- Each node acts both as a master and a slave for different shards.
- Writes (id = 2) to Shard A are regulated by Node A, as it is the master for Shard A.
- Node A replicates data (id = 2) to Node B, which is a slave for Shard A.

• Reads (id = 4) can be served directly by either Node B or Node C as they each contain Shard B.

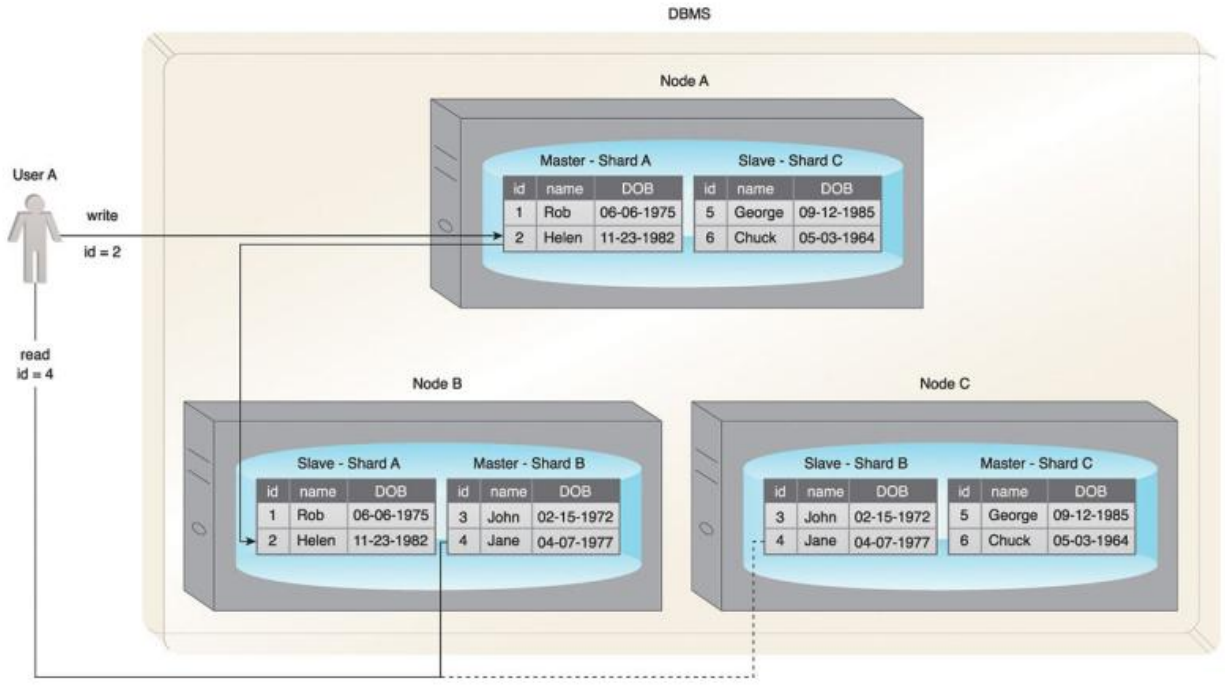


Figure 3.13 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches. [26]

Combining Sharding and Peer-to-Peer Replication

When combining sharding with peer-to-peer replication, each shard is replicated to multiple peers, and each peer is only responsible for a subset of the overall dataset.

Collectively, this helps achieve increased scalability and fault tolerance. As there is no master involved, there is no single point of failure and fault-tolerance for both read and write operations is supported.

In Figure 3.14:

• Each node contains replicas of two different shards.

• Writes (id = 3) are replicated to both Node A and Node C (Peers) as they are responsible for Shard C.

• Reads (id = 6) can be served by either Node B or Node C as they each contain Shard B.

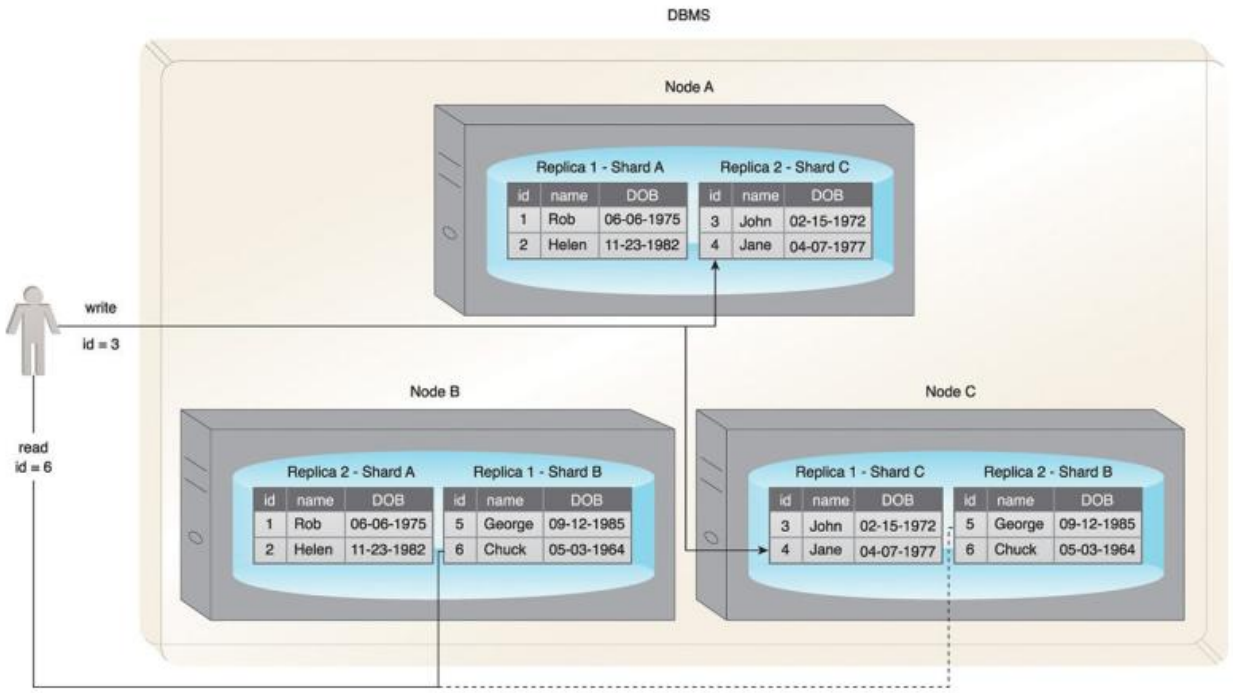


Figure 3.14 An example of the combination of sharding and peer-to-peer replication [26]

3.1.7 CAP Theorem [26]

The Consistency, Availability, and Partition tolerance (CAP) theorem, also known as Brewer's theorem, expresses a triple constraint related to distributed database systems. It states that a distributed database system, running on a cluster, can only provide two of the following three properties:

• ***Consistency*** – A read from any node results in the same data across multiple nodes (Figure 3.15)

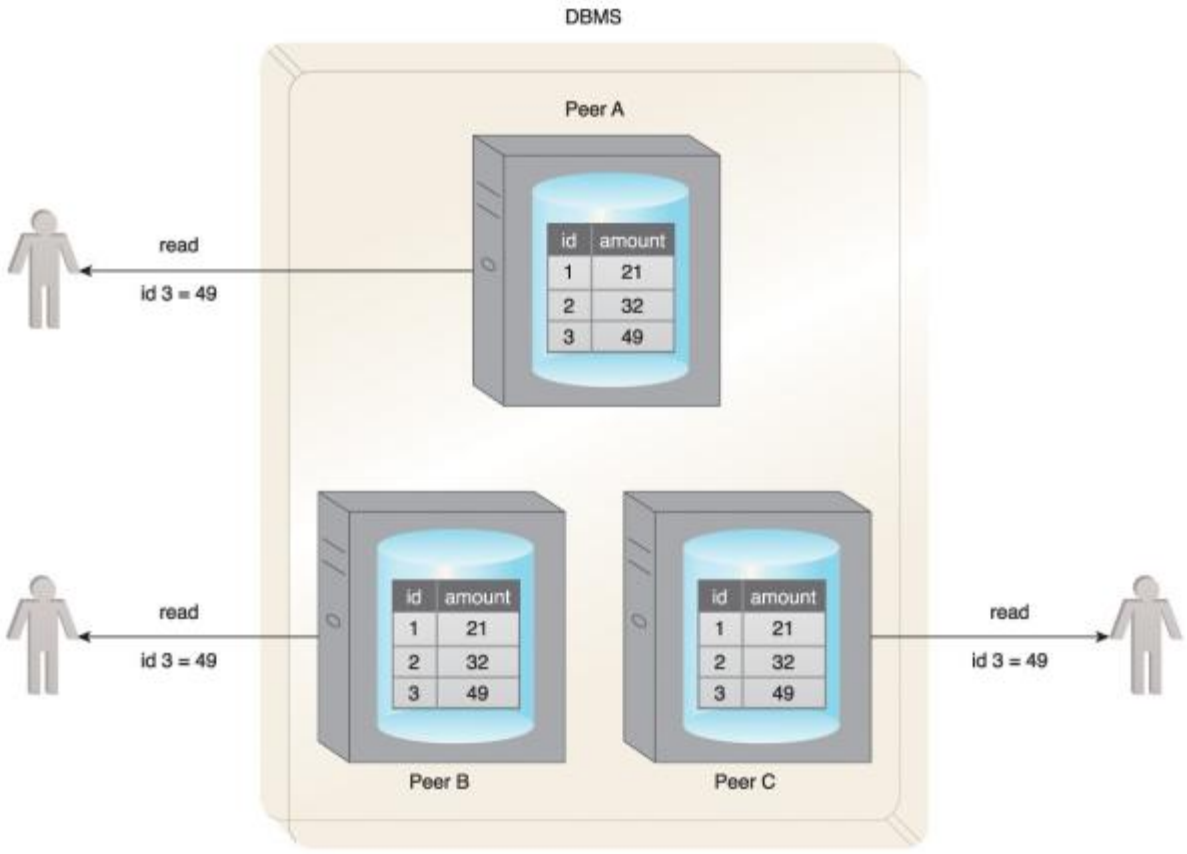


Figure 3.15 Consistency: all three users get the same value for the amount column even though three different nodes are serving the record. [26]

• ***Availability*** – A read/write request will always be acknowledged in the form of a success or a failure (Figure 3.16).

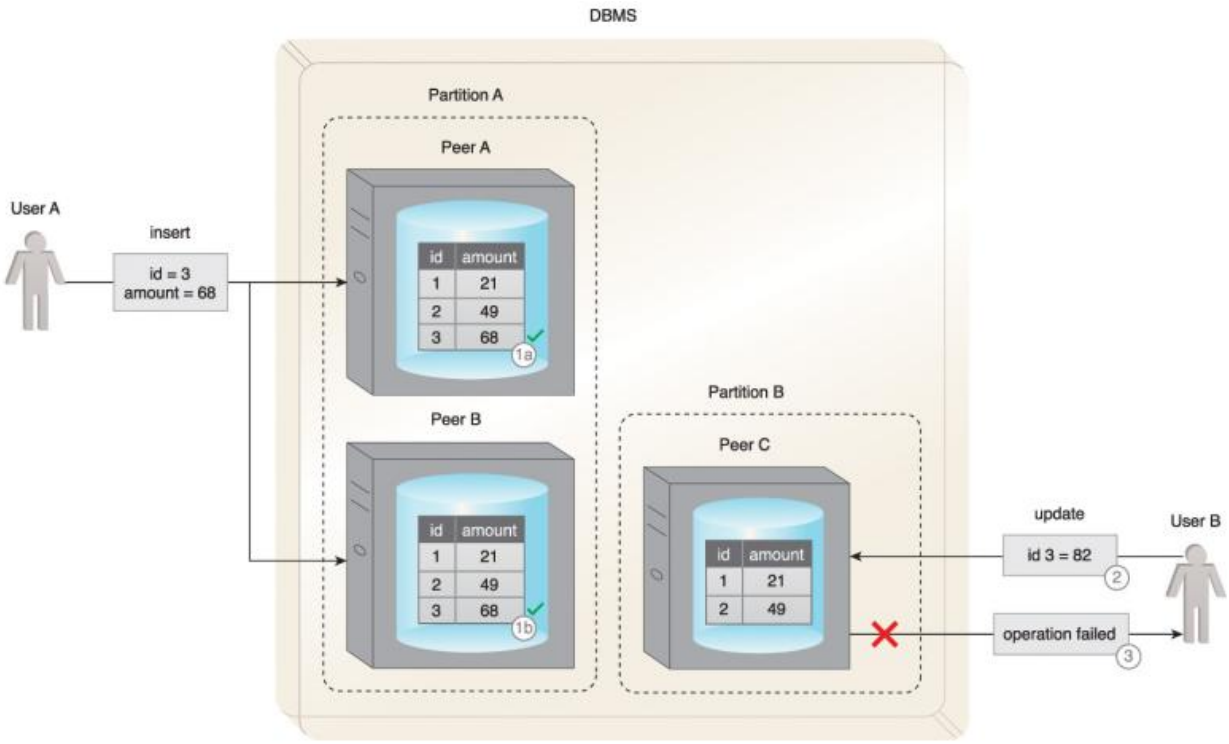


Figure 3.16 Availability and partition tolerance: in the event of a communication failure, requests from both users are still serviced (1, 2). However, with User B, the update fails as the record with id = 3 has not been copied over to Peer C. The user is duly notified (3) that the update has failed. [26]

• ***Partition tolerance*** – The database system can tolerate communication outages that split the cluster into multiple silos and can still service read/write requests (Figure 3.16).

The following scenarios demonstrate why only two of the three properties of the CAP theorem are simultaneously supportable. To aid this discussion, Figure 3.17 provides a Venn diagram showing the areas of overlap between consistency, availability and partition tolerance.

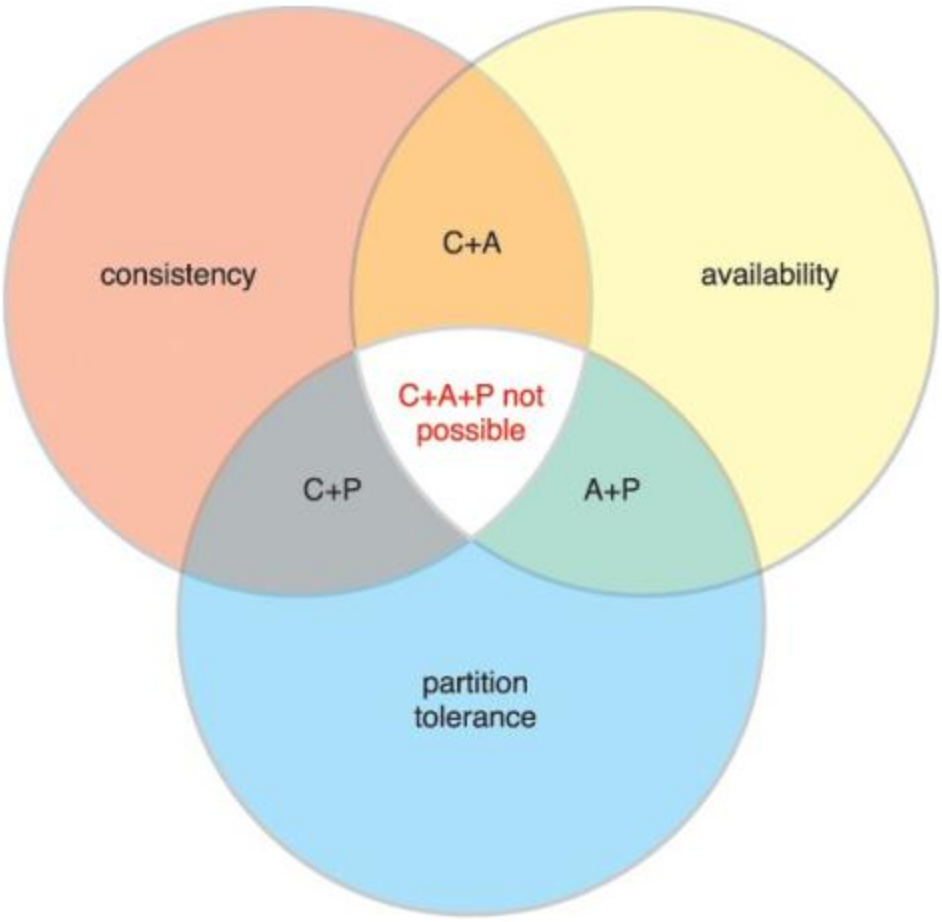


Figure 3.17 A Venn diagram summarizing the CAP theorem. [26]

If consistency (C) and availability (A) are required, available nodes need to communicate to ensure consistency (C). Therefore, partition tolerance (P) is not possible.
If consistency (C) and partition tolerance (P) are required, nodes cannot remain available (A) as the nodes will become unavailable while achieving a state of consistency (C).
If availability (A) and partition tolerance (P) are required, then consistency (C) is not possible because of the data communication requirement between the nodes. So, the database can remain available (A) but with inconsistent results.
In a distributed database, scalability and fault tolerance can be improved through additional nodes, although this challenges consistency (C). The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
Distributed database systems cannot be 100% partition tolerant (P). Although communication outages are rare and temporary, partition tolerance (P) must always be supported by a distributed database; therefore, CAP is generally a choice between choosing either C+P or A+P. The requirements of the system will dictate which is chosen.

3.1.8 ACID [26]

ACID is a database design principle related to transaction management. It is an acronym that stands for:

- atomicity
- consistency
- isolation
- durability

ACID is a transaction management style that leverages pessimistic concurrency controls to ensure consistency is maintained through the application of record locks. ACID is the traditional approach to database transaction management as it is leveraged by relational database management systems.

Atomicity ensures that all operations will always succeed or fail completely. In other words, there are no partial transactions.

The following steps are illustrated in Figure 3.18:

1. A user attempts to update three records as a part of a transaction.
2. Two records are successfully updated before the occurrence of an error.
3. As a result, the database roll backs any partial effects of the transaction and puts the system back to its prior state.

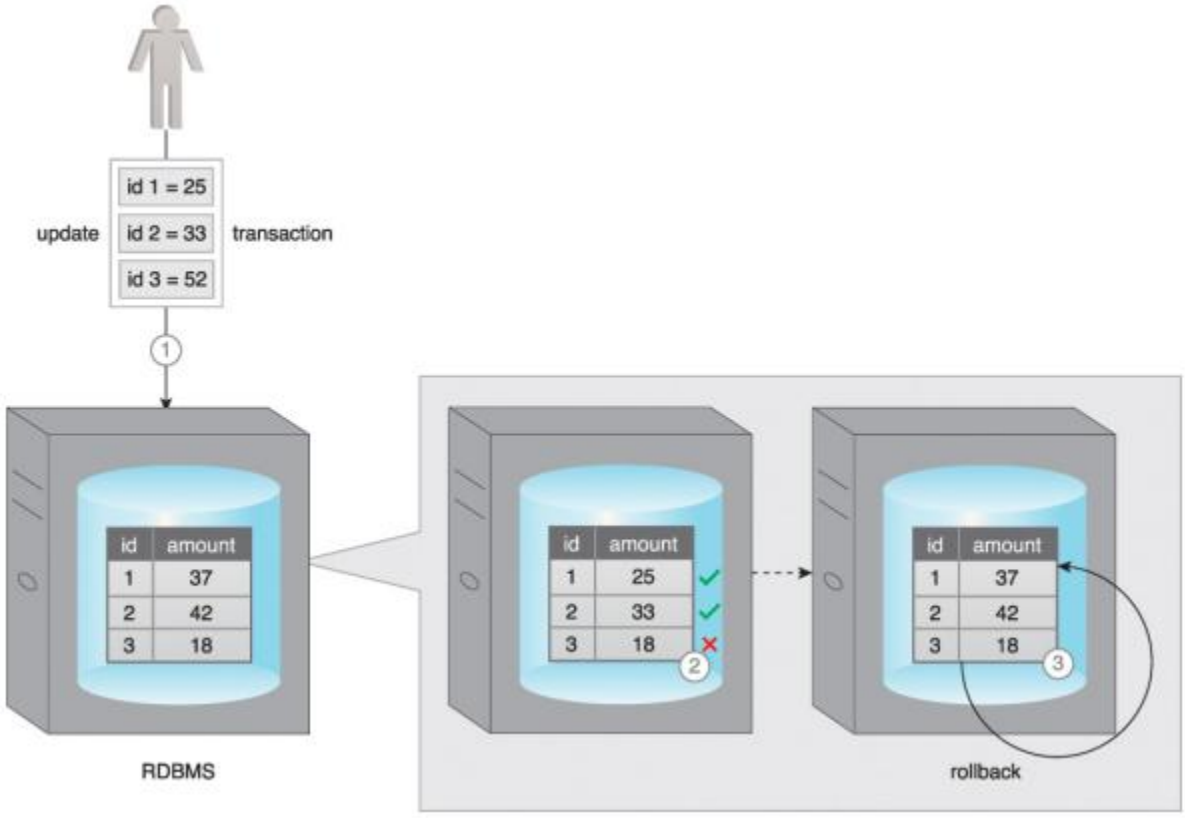


Figure 3.18 An example of the atomicity property of ACID is evident here. [26]

Consistency ensures that the database will always remain in a consistent state by ensuring that only data that conforms to the constraints of the database schema can be written to the database. Thus a database that is in a consistent state will remain in a consistent state following a successful transaction.

In Figure 3.19:

1. A user attempts to update the amount column of the table that is of type float with a varchar value.
2. The database applies its validation check and rejects this update because the value violates the constraint checks for the amount column.

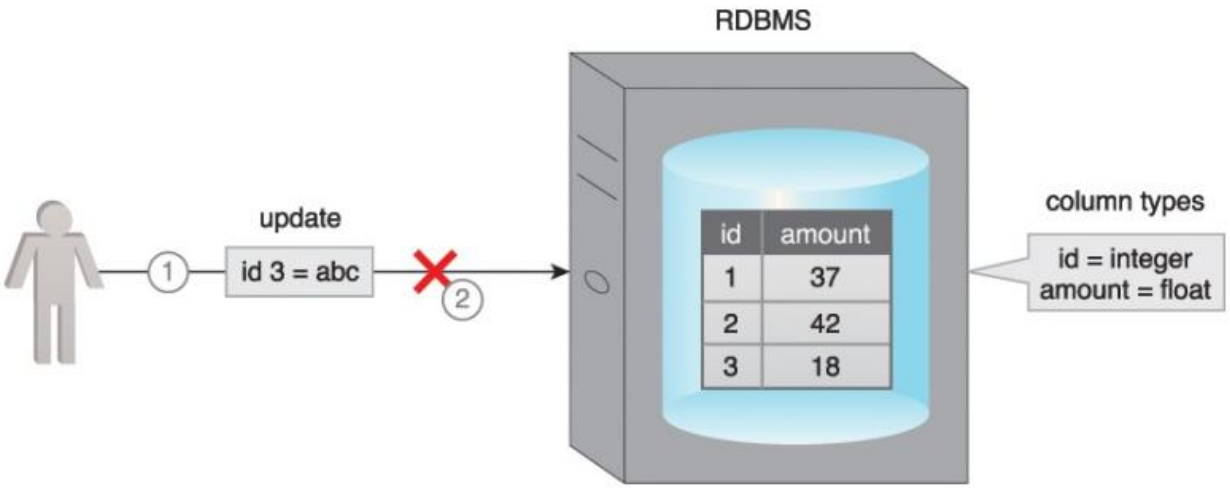


Figure 3.19 An example of the consistency of ACID [26]

Isolation ensures that the results of a transaction are not visible to other operations until it is complete.

In Figure 3.20:

1. User A attempts to update two records as part of a transaction.
2. The database successfully updates the first record.
3. However, before it can update the second record, User B attempts to update the same record. The database does not permit User B's update until User A's update succeeds or fails in full. This occurs because the record with id3 is locked by the database until the transaction is complete.

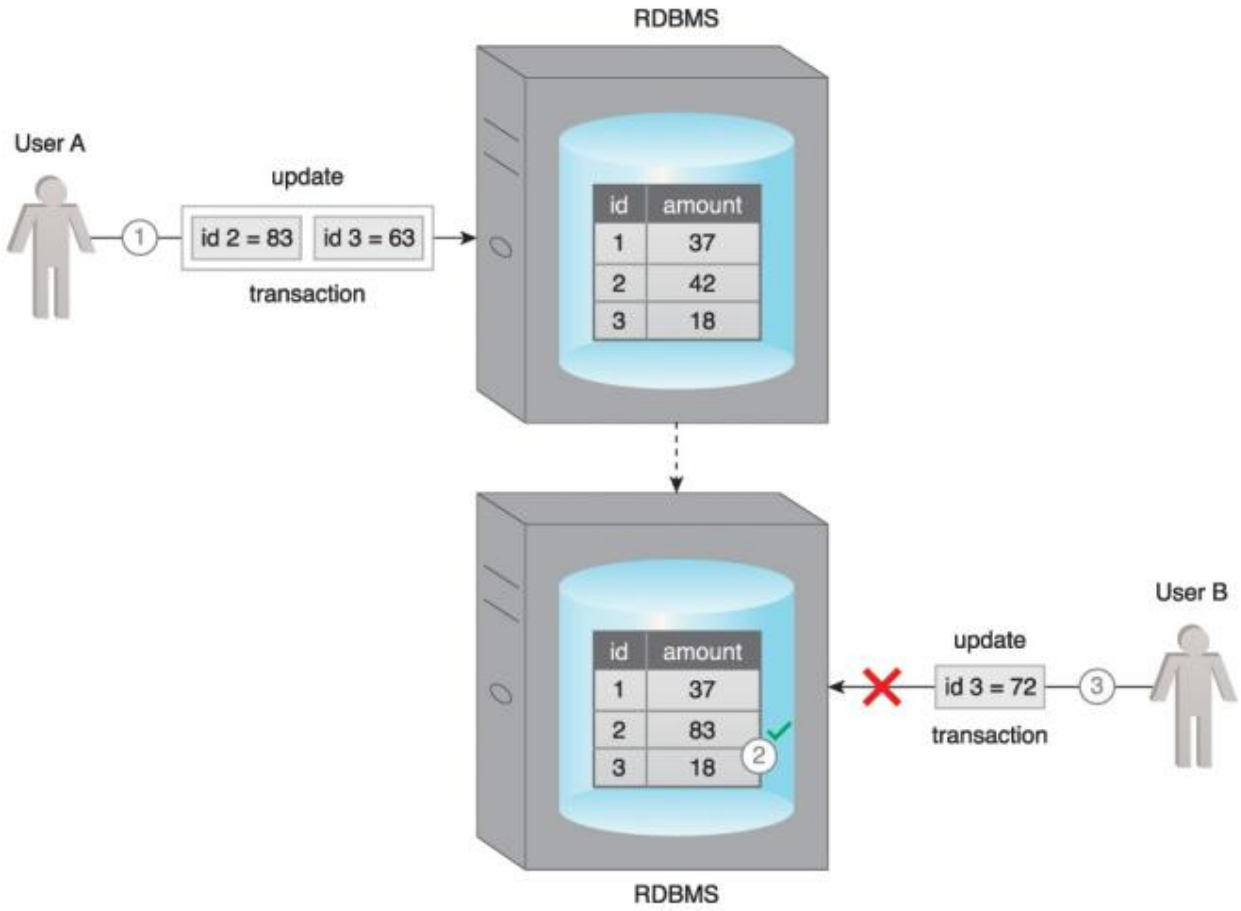


Figure 3.20 An example of the isolation property of ACID [26]

Durability ensures that the results of an operation are permanent. In other words, once a transaction has been committed, it cannot be rolled back. This is irrespective of any system failure.

In Figure 3.21:

1. A user updates a record as part of a transaction.
2. The database successfully updates the record.
3. Right after this update, a power failure occurs. The database maintains its state while there is no power.
4. The power is resumed.
5. The database serves the record as per last update when requested by the user.

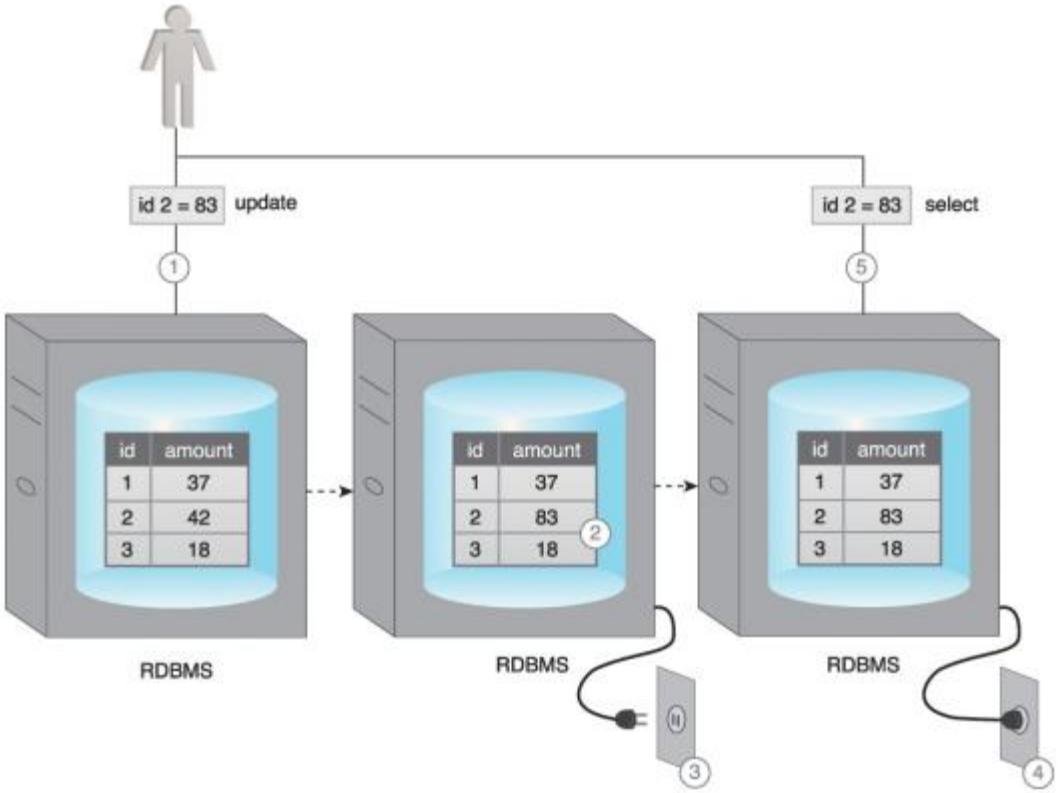


Figure 3.21 The durability characteristic of ACID. [26]

Figure 3.22 shows the results of the application of the ACID principle:
1. User A attempts to update a record as part of a transaction.
2. The database validates the value and the update is successfully applied.
3. After the successful completion of the transaction, when Users B and C request the same record, the database provides the updated value to both the users.

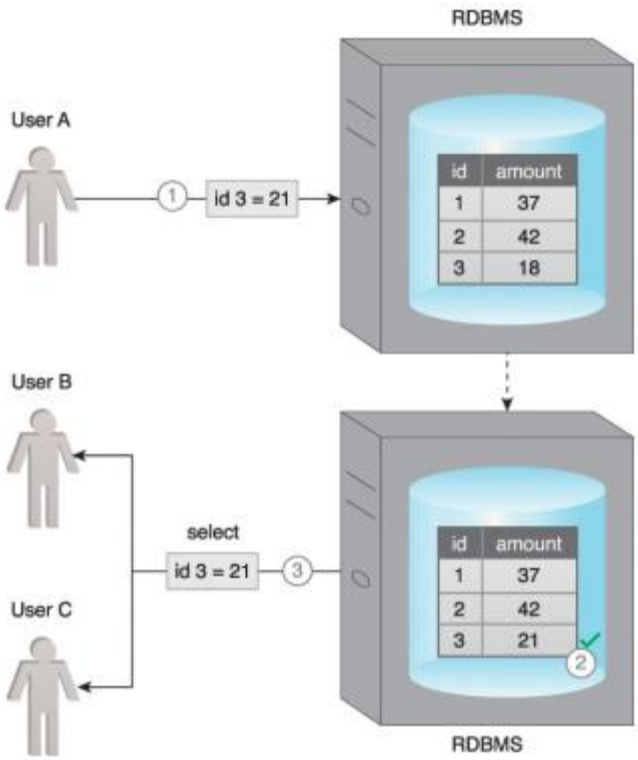


Figure 3.22 The ACID principle results in consistent database behavior. [26]

3.1.9 BASE [26]

BASE is a database design principle based on the CAP theorem and leveraged by database systems that use distributed technology. BASE stands for:

- basically available
- soft state
- eventual consistency

When a database supports BASE, it favors availability over consistency. In other words, the database is A+P from a CAP perspective. In essence, BASE leverages optimistic concurrency by relaxing the strong consistency constraints mandated by the ACID properties.

If a database is "basically available," that database will always acknowledge a client's request, either in the form of the requested data or a success/failure notification.

In Figure 3.23, the database is basically available, even though it has been partitioned as a result of a network failure.

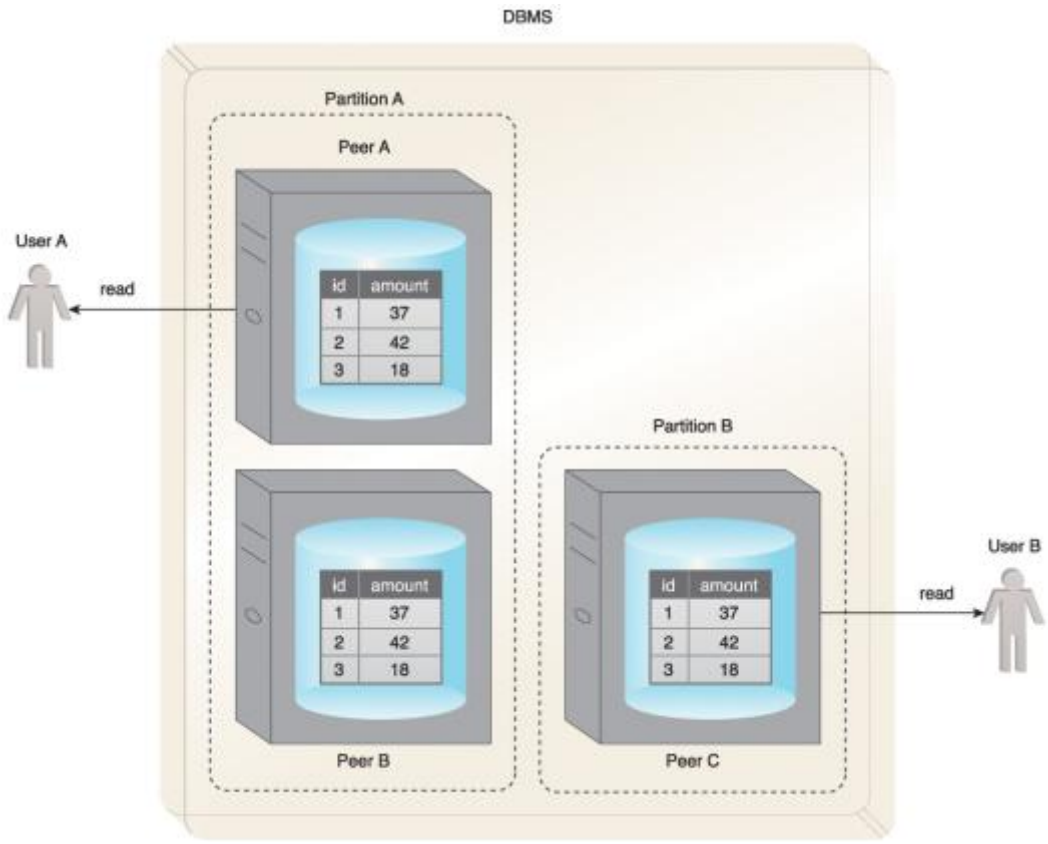


Figure 3.23 User A and User B receive data despite the database being partitioned by a network failure. [26]

Soft state means that a database may be in an inconsistent state when data is read; thus, the results may change if the same data is requested again. This is because the data could be updated for consistency, even though no user has written to the database between the two reads. This property is closely related to eventual consistency.

In Figure 3.24:

1. User A updates a record on Peer A.
2. Before the other peers are updated, User B requests the same record from Peer C.
3. The database is now in a soft state, and stale data is returned to User B.

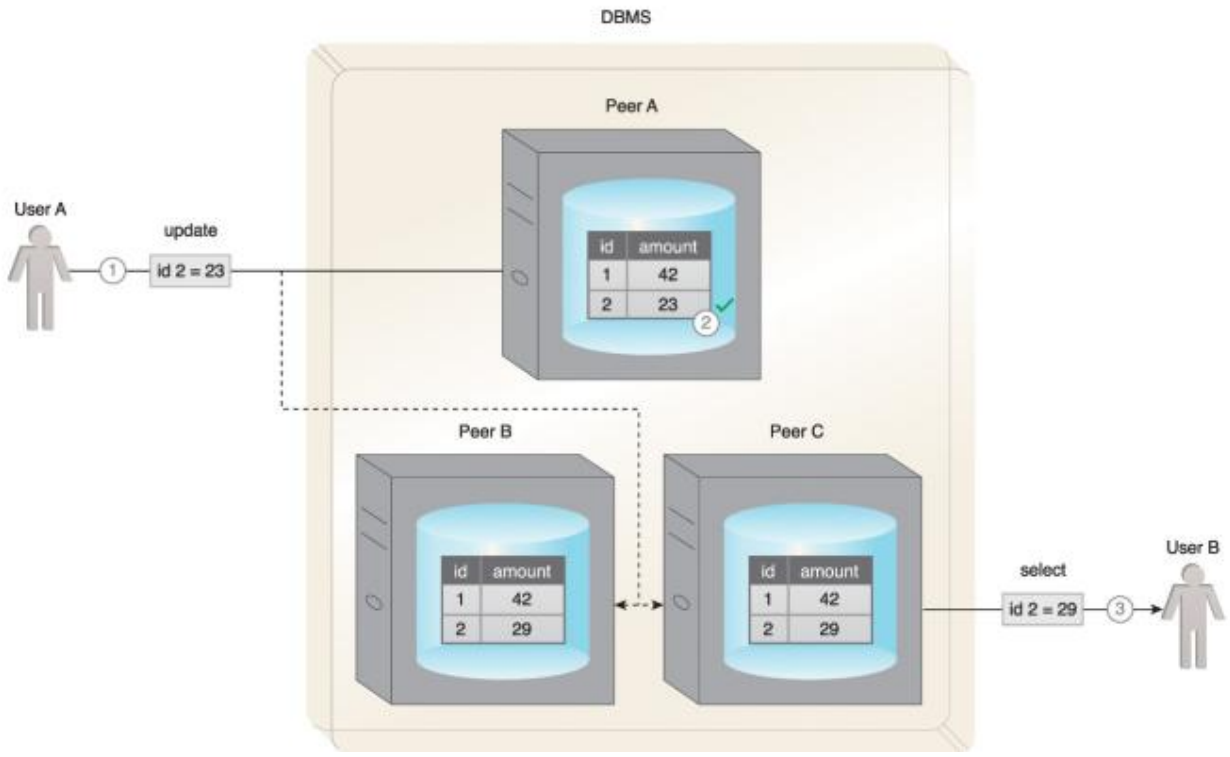


Figure 3.24 An example of the soft state property of BASE is shown here. [26]

Eventual consistency is the state in which reads by different clients, immediately following a write to the database, may not return consistent results. The database only attains consistency once the changes have been propagated to all nodes. While the database is in the process of attaining the state of eventual consistency, it will be in a soft state.

In Figure 3.25:

1. User A updates a record.
2. The record only gets updated at Peer A, but before the other peers can be updated, User B requests the same record.
3. The database is now in a soft state. Stale data is returned to User B from Peer C.
4. However, the consistency is eventually attained, and User C gets the correct value.

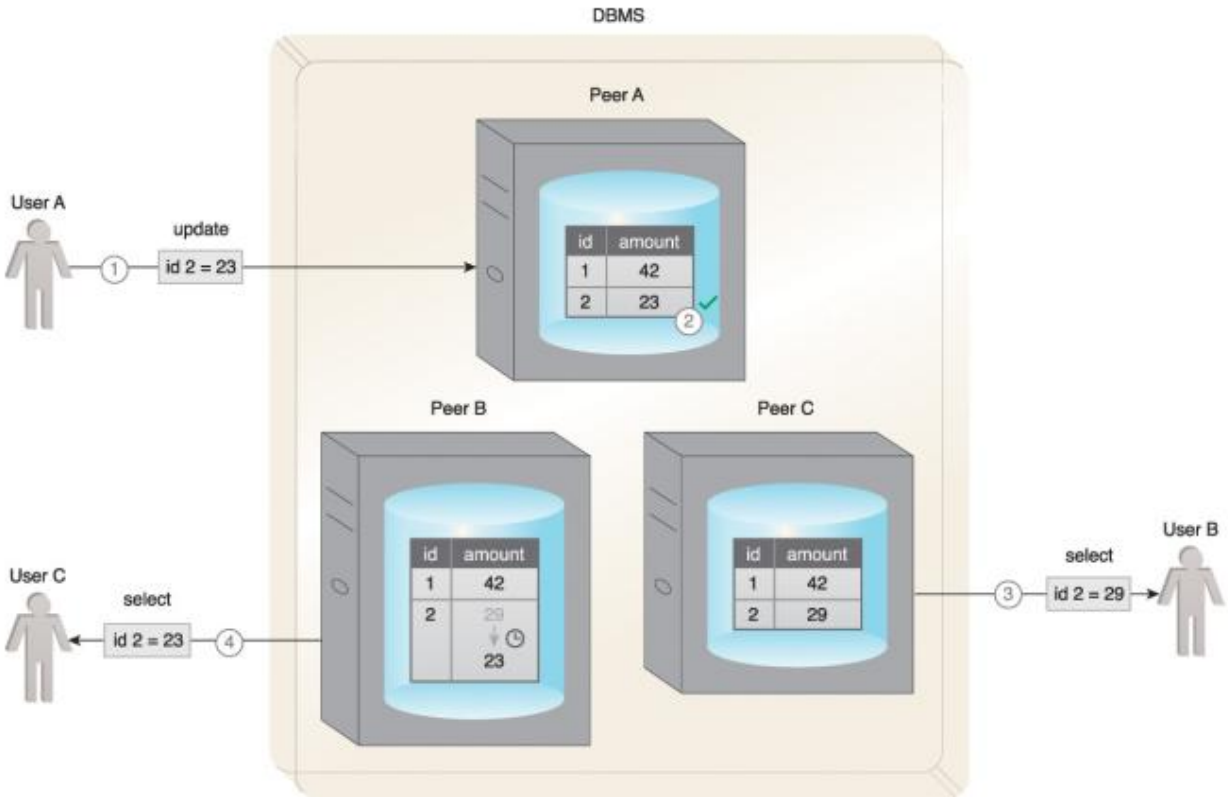


Figure 3.25 An example of the eventual consistency property of BASE. [26]

BASE emphasizes availability over immediate consistency, in contrast to ACID, which ensures immediate consistency at the expense of availability due to record locking. This soft approach toward consistency allows BASE compliant databases to serve multiple clients without any latency albeit serving inconsistent results. However, BASE-compliant databases are not useful for transactional systems where lack of consistency is a concern.

3.1.10 Case Study Example [26]

ETI's IT environment currently utilizes both Linux and Windows operating systems. Consequently, both ext and NTFS file systems are in use. The webserversand some of the application servers employ ext, while the rest of the application servers, the database servers and the end-users' PCs are configured to use NTFS. Network-attached storage (NAS) configured with RAID 5 is also used for fault tolerant document storage. Although the IT team is conversant with file systems, the concepts of cluster, distributed file system and NoSQL are new to the group. Nevertheless, after a discussion with the trained IT team members, the entire group is able to understand these concepts and technologies.
ETI's current IT landscape comprises entirely of relational databases that employ the ACID database design principle. The IT team has no understanding of the BASE principle and is having trouble comprehending the CAP theorem. Some of the team

members are unsure about the need and importance of these concepts with regards to Big Data dataset storage. Seeing this, the IT-trained employees try to ease their fellow team members' confusion by explaining that these concepts are only applicable to the storage of enormous amounts of data in a distributed fashion on a cluster. Clusters have become the obvious choice for storing very large volume of data due to their ability to support linear scalability by scaling out.

Since clusters are comprised of nodes connected via a network, communication failures that create silos or partitions of a cluster are inevitable. To address the partition issue, the BASE principle and CAP theorem are introduced. They further explain that any database following the BASE principle becomes more responsive to its clients, albeit the data being read may be inconsistent when compared to a database that follows the ACID principle. Having understood the BASE principle, the IT team more easily comprehends why a database implemented in a cluster has to choose between consistency and availability.

Although none of the existing relational databases use sharding, almost all relational databases are replicated for disaster recovery and operational reporting. To better understand the concepts of sharding and replication, the IT team goes through an exercise of how these concepts can be applied to the insurance quote data as a large number of quotes are created and accessed quickly. For sharding, the team believes that using the type (the insurance sector—heath, building, marine and aviation) of the insurance quote as sharding criteria will create a balanced set of data across multiple nodes, for queries are mostly executed within the same insurance sector, and inter-sector queries are rare. With regards to replication, the team is in favor of choosing a NoSQL database that implements the peer-to-peer replication strategy. The reason behind their decision is that the insurance quotes are created and retrieved quite frequently but seldom updated. Hence the chances of getting an inconsistent record are low. Considering this, the team favors read/write performance over consistency by choosing peer-to-peer replication.

3.2 APACHE MAHOUT [27, 30]

Mahout is an open source *machine learning* library from Apache. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. This can mean many things, but at the moment for Mahout it means primarily recommender engines (collaborative filtering), clustering, and classification.

It's also *scalable*. Mahout aims to be the machine learning tool of choice when the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, and some portions are built upon Apache's Hadoop distributed computation project.

Finally, it's a Java library. It doesn't provide a user interface, a prepackaged server, or an installer. It's a framework of tools intended to be used and adapted by developers.
To set the stage, this chapter will take a brief look at the sorts of machine learning that Mahout can help you perform on your data—using recommender engines, clustering, and classification—by looking at some familiar real-world instances.

Features of Mahout [30]
The primitive features of Apache Mahout are listed below.

- The algorithms of Mahout are written on top of Hadoop, so it works well in distributed environment. Mahout uses the Apache Hadoop library to scale effectively in the cloud.
- Mahout offers the coder a ready-to-use framework for doing data mining tasks on large volumes of data.
- Mahout lets applications to analyze large sets of data effectively and in quick time.
- Includes several MapReduce enabled clustering implementations such as k-means, fuzzy k-means, Canopy, Dirichlet, and Mean-Shift.
- Supports Distributed Naive Bayes and Complementary Naive Bayes classification implementations.
- Comes with distributed fitness function capabilities for evolutionary programming.
- Includes matrix and vector libraries.

3.2.1 Mahout's Story [27]

First, some background on Mahout itself is in order. You may be wondering how to pronounce Mahout: in the way it's commonly Anglicized, it should rhyme with trout. It's a Hindi word that refers to an elephant driver, and to explain that one, here's a little history.
Mahout began life in 2008 as a subproject of Apache's Lucene project, which provides the well-known open source search engine of the same name. Lucene provides advanced implementations of search, text mining, and information-retrieval techniques. In the universe of computer science, these concepts are adjacent to machine learning techniques like clustering and, to an extent, classification. As a result, some of the work of the Lucene committers that fell more into these machine learning areas was spun off into its

own subproject. Soon after, Mahout absorbed the Taste open source collaborative filtering project.

Figure 3.26 shows some of Mahout's lineage within the Apache Software Foundation. As of April 2010, Mahout became a top-level Apache project in its own right, and got a brand-new elephant rider logo to boot.

Much of Mahout's work has been not only implementing these algorithms conventionally, in an efficient and scalable way, but also converting some of these algorithms to work at scale on top of Hadoop. Hadoop's mascot is an elephant, which at last explains the project name!

Mahout incubates a number of techniques and algorithms, many still in development or in an experimental phase (https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms). At this early stage in the project's life, three core themes are evident: recommender engines (collaborative filtering), clustering, and classification.

This is by no means all that exists within Mahout, but they are the most prominent and mature themes at the time of writing.

Figure 3.27 Apache Mahout and its related projects within the Apache Software Foundation. [27]

3.2.2 What is Machine Learning? [30]

Machine learning is a branch of science that deals with programming the systems in such a way that they automatically learn and improve with experience. Here, learning means recognizing and understanding the input data and making wise decisions based on the supplied data.

It is very difficult to cater to all the decisions based on all possible inputs. To tackle this problem, algorithms are developed. These algorithms build knowledge from specific data and past experience with the principles of statistics, probability theory, logic, combinatorial optimization, search, reinforcement learning, and control theory.

The developed algorithms form the basis of various applications such as:

- Vision processing
- Language processing
- Forecasting (e.g., stock market trends)
- Pattern recognition
- Games
- Data mining
- Expert systems
- Robotics

Machine learning is a vast area and it is quite beyond the scope of this tutorial to cover all its features. There are several ways to implement machine learning techniques, however the most commonly used ones are supervised and unsupervised learning.

Supervised Learning [30]
Supervised learning deals with learning a function from available training data. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. Common examples of supervised learning include:

- classifying e-mails as spam,
- labeling webpages based on their content, and
- voice recognition.

There are many supervised learning algorithms such as neural networks, Support Vector Machines (SVMs), and Naive Bayes classifiers. Mahout implements Naive Bayes classifier.

Unsupervised Learning [30]

Unsupervised learning makes sense of unlabeled data without having any predefined dataset for its training. Unsupervised learning is an extremely powerful tool for analyzing available data and look for patterns and trends. It is most commonly used for clustering similar input into logical groups. Common approaches to unsupervised learning include:

- k-means
- self-organizing maps, and
- hierarchical clustering

Recommendation [30]

Recommendation is a popular technique that provides close recommendations based on user information such as previous purchases, clicks, and ratings.

Amazon uses this technique to display a list of recommended items that you might be interested in, drawing information from your past actions. There are recommender engines that work behind Amazon to capture user behavior and recommend selected items based on your earlier actions.

Facebook uses the recommender technique to identify and recommend the "people you may know list".

Classification [30]

Classification, also known as categorization, is a machine learning technique that uses known data to determine how the new data should be classified into a set of existing categories. Classification is a form of supervised learning.

Mail service providers such as Yahoo! and Gmail use this technique to decide whether a new mail should be classified as a spam. The categorization algorithm trains itself by analyzing user habits of marking certain mails as spams. Based on that, the classifier decides whether a future mail should be deposited in your inbox or in the spams folder.

iTunes application uses classification to prepare playlists.

Clustering [30]

Clustering is used to form groups or clusters of similar data based on common characteristics. Clustering is a form of unsupervised learning.

Search engines such as Google and Yahoo! use clustering techniques to group data with similar characteristics.

Newsgroups use clustering techniques to group various articles based on related topics.

The clustering engine goes through the input data completely and based on the characteristics of the data, it will decide under which cluster it should be grouped. Take a look at the following example.

3.2.3 Mahout's Machine Learning Themes [27, 30]

Although Mahout is, in theory, a project open to implementations of all kinds of machine learning techniques, it's in practice a project that focuses on three key areas of machine learning at the moment. These are recommender engines (collaborative filtering), clustering, and classification.

3.2.3.1 Recommender engines [27]

Recommender engines are the most immediately recognizable machine learning technique in use today. You'll have seen services or sites that attempt to recommend books or movies or articles based on your past actions. They try to infer tastes and preferences and identify unknown items that are of interest:

- Amazon.com is perhaps the most famous e-commerce site to deploy recommendations. Based on purchases and site activity, Amazon recommends books and other items likely to be of interest. See figure 3.28.
- Netflix similarly recommends DVDs that may be of interest, and famously offered a $1,000,000 prize to researchers who could improve the quality of their recommendations.
- Dating sites like Líbímseti can even recommend people to people.
- Social networking sites like Facebook use variants on recommender techniques to identify people most likely to be as-yet-unconnected friends.

As Amazon and others have demonstrated, recommenders can have concrete commercial value by enabling smart cross-selling opportunities. One firm reports that recommending products to users can drive an 8 to 12 percent increase in sales.[28]

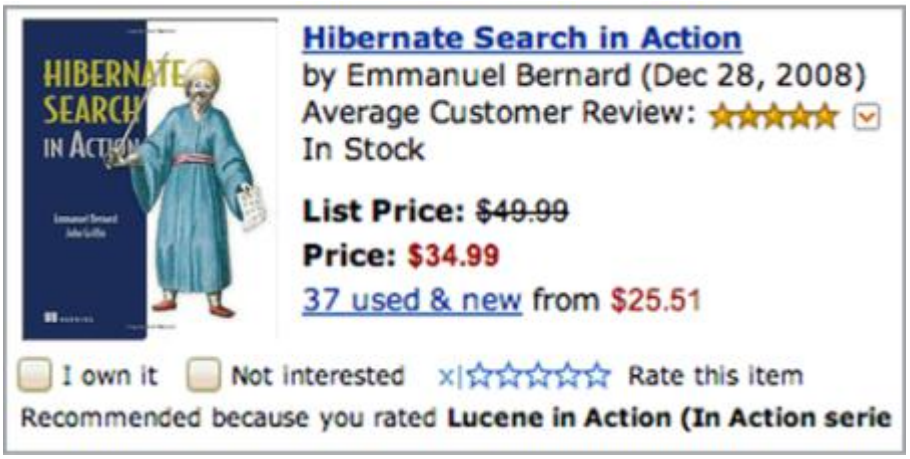


Figure 3.28 A recommendation from Amazon. Based on past purchase history and other activity of customers like the user, Amazon considers this to be something the user is interested in. It can even list similar items that the user has bought or liked that in part caused the recommendation. [27]

Suppose you want to purchase the book "Mahout in Action" from Amazon. Along with the selected product, Amazon also displays a list of related recommended items, as shown below.

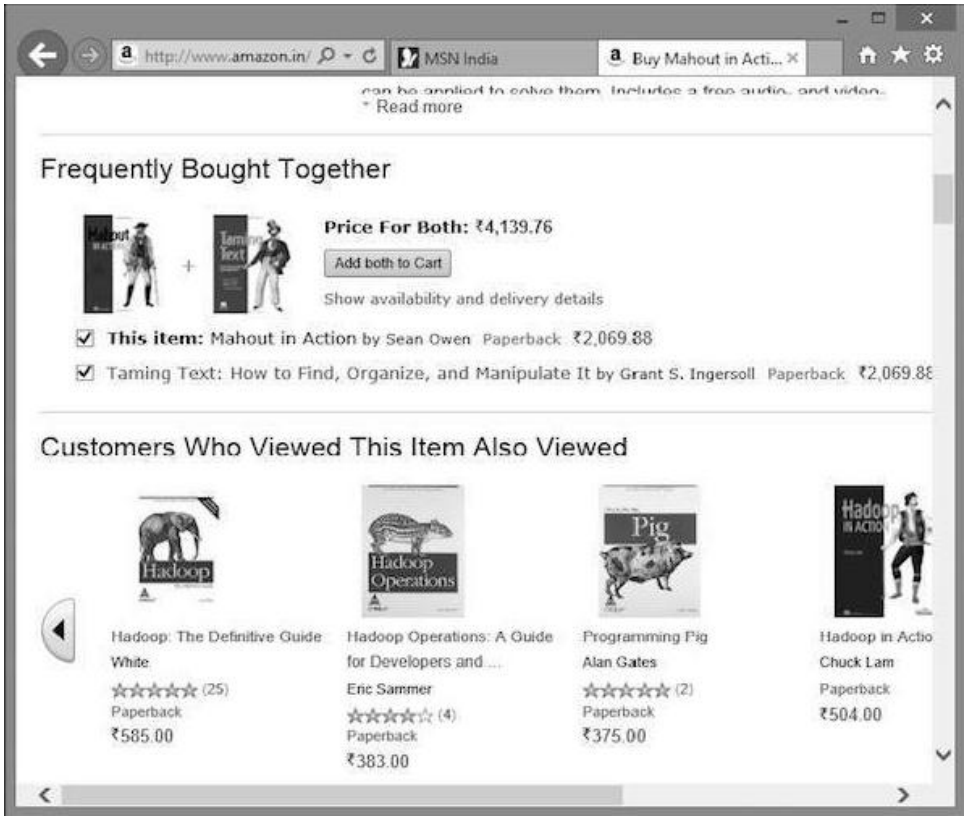


Figure 3.29 [30]

Such recommendation lists are produced with the help of recommender engines. Mahout provides recommender engines of several types such as:

- user-based recommenders,
- item-based recommenders, and
- several other algorithms.
- Mahout Recommender Engine

Mahout has a non-distributed, non-Hadoop-based recommender engine. You should pass a text document having user preferences for items. And the output of this engine would be the estimated preferences of a particular user for other items.

Example [30]
Consider a website that sells consumer goods such as mobiles, gadgets, and their accessories. If we want to implement the features of Mahout in such a site, then we can build a recommender engine. This engine analyzes past purchase data of the users and recommends new products based on that.

The components provided by Mahout to build a recommender engine are as follows:

- DataModel
- UserSimilarity
- ItemSimilarity
- UserNeighborhood
- Recommender

From the data store, the data model is prepared and is passed as an input to the recommender engine. The Recommender engine generates the recommendations for a particular user. Given below is the architecture of recommender engine.

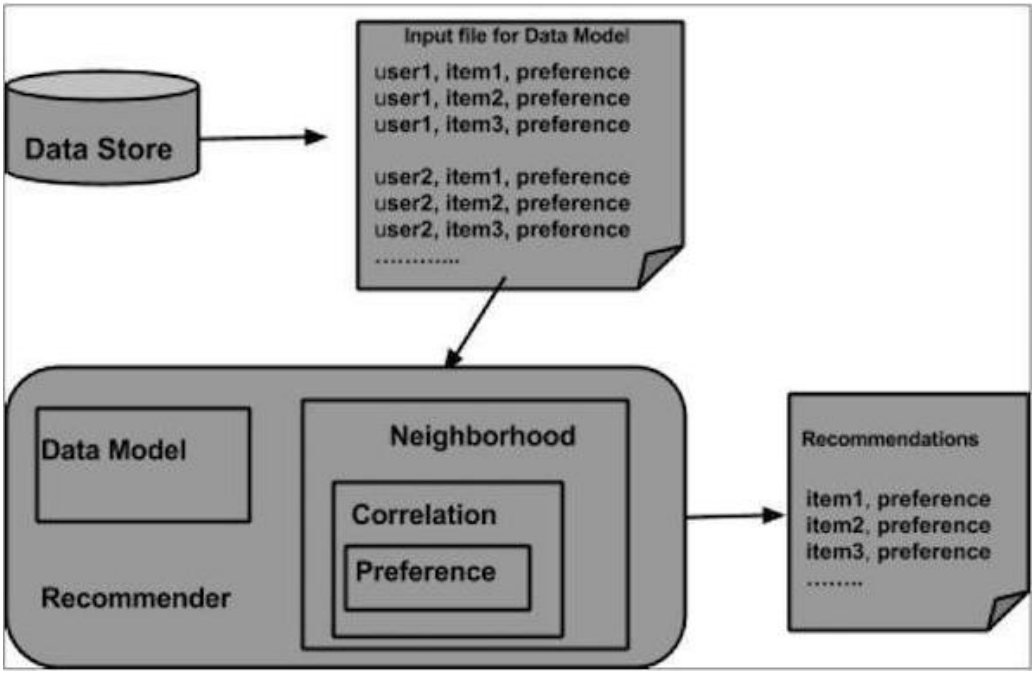


Figure 3.30 Architecture of Recommender Engine [30]

Building a Recommender using Mahout [30]

Here are the steps to develop a simple recommender:

Step1: Create DataModel Object

The constructor of **PearsonCorrelationSimilarity** class requires a data model object, which holds a file that contains the Users, Items, and Preferences details of a product. Here is the sample data model file:

```
1,00,1.0
1,01,2.0
1,02,5.0
1,03,5.0
1,04,5.0

2,00,1.0
2,01,2.0
2,05,5.0
2,06,4.5
2,02,5.0

3,01,2.5
3,02,5.0
3,03,4.0
3,04,3.0

4,00,5.0
4,01,5.0
4,02,5.0
4,03,0.0
```

The **DataModel** object requires the file object, which contains the path of the input file. Create the **DataModel** object as shown below.

```
DataModel datamodel = new FileDataModel(new File("input file"));
```

Step2: Create UserSimilarity Object

Create **UserSimilarity** object using PearsonCorrelationSimilarity class as shown below:

```
UserSimilarity similarity = new PearsonCorrelationSimilarity(datamodel);
```

Step3: Create UserNeighborhood object

This object computes a "neighborhood" of users like a given user. There are two types of neighborhoods:

- **NearestNUserNeighborhood** - This class computes a neighborhood consisting of the nearest n users to a given user. "Nearest" is defined by the given UserSimilarity.

- **ThresholdUserNeighborhood** - This class computes a neighborhood consisting of all the users whose similarity to the given user meets or exceeds a certain threshold. Similarity is defined by the given UserSimilarity.

Here we are using **ThresholdUserNeighborhood** and set the limit of preference to 3.0.

```
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(3.0, similarity, model);
```

Step4: Create Recommender Object

Create **UserbasedRecomender** object. Pass all the above created objects to its constructor as shown below.

```
UserBasedRecommender recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);
```

Step5: Recommend Items to a User

Recommend products to a user using the recommend() method of **Recommender** interface. This method requires two parameters. The first represents the user id of the user to whom we need to send the recommendations, and the second represents the number of recommendations to be sent. Here is the usage of **recommender()** method:

```
List<RecommendedItem> recommendations = recommender.recommend(2, 3);

for (RecommendedItem recommendation : recommendations) {
   System.out.println(recommendation);
 }
```

Example Program [30]

Given below is an example program to set recommendation. Prepare the recommendations for the user with user id 2.

```
import java.io.File;
import java.util.List;

import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;

import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;

import org.apache.mahout.cf.taste.similarity.UserSimilarity;

public class Recommender {
  public static void main(String args[]){
    try{
      //Creating data model
      DataModel datamodel = new FileDataModel(new File("data")); //data

      //Creating UserSimilarity object.
      UserSimilarity usersimilarity = new PearsonCorrelationSimilarity(datamodel);
```

```
        //Creating UserNeighbourHHood object.

        UserNeighborhood userneighborhood = new ThresholdUserNeighborhood(3.0,
usersimilarity, datamodel);



        //Create UserRecomender

        UserBasedRecommender recommender = new
GenericUserBasedRecommender(datamodel, userneighborhood, usersimilarity);



        List<RecommendedItem> recommendations = recommender.recommend(2, 3);



        for (RecommendedItem recommendation : recommendations) {

          System.out.println(recommendation);

        }


     }catch(Exception e){}


   }

  }
```

Compile the program using the following commands:

```
javac Recommender.java
java Recommender
```

It should produce the following output:

```
RecommendedItem [item:3, value:4.5]
RecommendedItem [item:4, value:4.0]
```

3.2.3.2 Clustering [30]

Clustering is less apparent, but it turns up in equally well-known contexts. As its name implies, clustering techniques attempt to group a large number of things together into clusters that share some similarity. It's a way to discover hierarchy and order in a large or hard-to-understand data set and in that way reveal interesting patterns or make the data set easier to comprehend.

- Google News groups news articles by topic using clustering techniques, in order to present news grouped by logical story, rather than presenting a raw listing of all articles. Figure 3.30 illustrates this.
- Search engines like Clusty group their search results for similar reasons.
- Consumers may be grouped into segments (clusters) using clustering techniques based on attributes like income, location, and buying habits.

Clustering helps identify structure, and even hierarchy, among a large collection of things that may be otherwise difficult to make sense of. Enterprises might use this technique to discover hidden groupings among users, or to organize a large collection of documents sensibly, or to discover common usage patterns for a site based on logs.

Figure 3.30 A sample news grouping from Google News. A detailed snippet from one representative story is displayed, and links to a few other similar stories within the cluster for this topic are shown. Links to all the stories that are clustered together in this topic are available too. [27]

Applications of Clustering

- Clustering is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.

- Clustering can help marketers discover distinct groups in their customer basis. And they can characterize their customer groups based on purchasing patterns.

- In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality and gain insight into structures inherent in populations.

- Clustering helps in identification of areas of similar land use in an earth observation database.

- Clustering also helps in classifying documents on the web for information discovery.

- Clustering is used in outlier detection applications such as detection of credit card fraud.

- As a data mining function, Cluster Analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

Using Mahout, we can cluster a given set of data. The steps required are as follows:

- **Algorithm** You need to select a suitable clustering algorithm to group the elements of a cluster.

- **Similarity and Dissimilarity** You need to have a rule in place to verify the similarity between the newly encountered elements and the elements in the groups.

- **Stopping Condition** A stopping condition is required to define the point where no clustering is required.

Procedure of Clustering

To cluster the given data you need to -

- Start the Hadoop server. Create required directories for storing files in Hadoop File System. (Create directories for input file, sequence file, and clustered output in case of canopy).

- Copy the input file to the Hadoop File system from Unix file system.

- Prepare the sequence file from the input data.
- Run any of the available clustering algorithms.
- Get the clustered data.

Starting Hadoop

Mahout works with Hadoop, hence make sure that the Hadoop server is up and running.

```
$ cd HADOOP_HOME/bin
$ start-all.sh
```

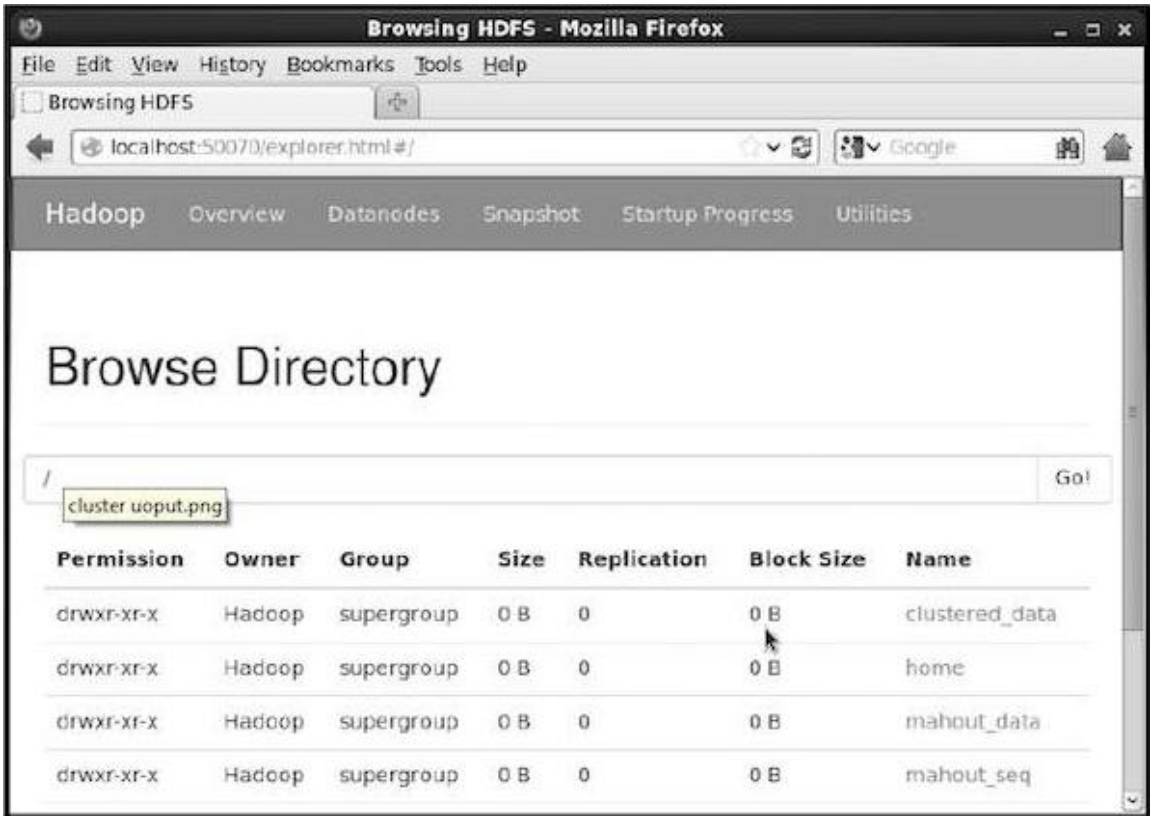
Preparing Input File Directories

Create directories in the Hadoop file system to store the input file, sequence files, and clustered data using the following command:

```
$ hadoop fs -p mkdir /mahout_data
$ hadoop fs -p mkdir /clustered_data
$ hadoop fs -p mkdir /mahout_seq
```

You can verify whether the directory is created using the hadoop web interface in the following URL - http://localhost:50070/

It gives you the output as shown below:



Copying Input File to HDFS

Now, copy the input data file from the Linux file system to mahout_data directory in the Hadoop File System as shown below. Assume your input file is mydata.txt and it is in the /home/Hadoop/data/ directory.

```
$ hadoop fs -put /home/Hadoop/data/mydata.txt /mahout_data/
```

Preparing the Sequence File
Mahout provides you a utility to convert the given input file in to a sequence file format. This utility requires two parameters.

- The input file directory where the original data resides.
- The output file directory where the clustered data is to be stored.

Given below is the help prompt of mahout **seqdirectory** utility.

Step 1: Browse to the Mahout home directory. You can get help of the utility as shown below:

```
[Hadoop@localhost bin]$ ./mahout seqdirectory --help
Job-Specific Options:
--input (-i) input Path to job input directory.
--output (-o) output The directory pathname for output.
--overwrite (-ow) If present, overwrite the output directory
```

Generate the sequence file using the utility using the following syntax:

```
mahout seqdirectory -i <input file path> -o <output directory>
```

Example

```
mahout seqdirectory
-i hdfs://localhost:9000/mahout_seq/
-o hdfs://localhost:9000/clustered_data/
```

Clustering Algorithms [30]

Mahout supports two main algorithms for clustering namely:

- Canopy clustering
- K-means clustering

Canopy Clustering [30]

Canopy clustering is a simple and fast technique used by Mahout for clustering purpose. The objects will be treated as points in a plain space. This technique is often used as an initial step in other clustering techniques such as k-means clustering. You can run a Canopy job using the following syntax:

```
mahout canopy -i <input vectors directory>
-o <output directory>
-t1 <threshold value 1>
-t2 <threshold value 2>
```

Canopy job requires an input file directory with the sequence file and an output directory where the clustered data is to be stored.

Example

```
mahout canopy -i hdfs://localhost:9000/mahout_seq/mydata.seq
-o hdfs://localhost:9000/clustered_data
-t1 20
-t2 30
```

You will get the clustered data generated in the given output directory.

K-means Clustering [30]

K-means clustering is an important clustering algorithm. The k in k-means clustering algorithm represents the number of clusters the data is to be divided into. For example, the k value specified to this algorithm is selected as 3, the algorithm is going to divide the data into 3 clusters.

Each object will be represented as vector in space. Initially k points will be chosen by the algorithm randomly and treated as centers, every object closest to each center are clustered. There are several algorithms for the distance measure and the user should choose the required one.

Creating Vector Files

- Unlike Canopy algorithm, the k-means algorithm requires vector files as input, therefore you have to create vector files.

- To generate vector files from sequence file format, Mahout provides the **seq2parse** utility.

Given below are some of the options of **seq2parse** utility. Create vector files using these options.

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName  The class name of the analyzer
--chunkSize (-chunk) chunkSize    The chunkSize in MegaBytes.
--output (-o) output              The directory pathname for o/p
--input (-i) input                Path to job input directory.
```

After creating vectors, proceed with k-means algorithm. The syntax to run k-means job is as follows:

```
mahout kmeans -i <input vectors directory>
-c  <input clusters directory>
-o  <output working directory>
-dm <Distance Measure technique>
-x  <maximum number of iterations>
-k  <number of initial clusters>
```

K-means clustering job requires input vector directory, output clusters directory, distance measure, maximum number of iterations to be carried out, and an integer value representing the number of clusters the input data is to be divided into.

3.2.3.3 Classification [30]

Classification techniques decide how much a thing is or isn't part of some type or category, or how much it does or doesn't have some attribute. Classification, like clustering, is ubiquitous, but it's even more behind the scenes. Often these systems learn by reviewing many instances of items in the categories in order to deduce classification rules. This general idea has many applications:

- Yahoo! Mail decides whether or not incoming messages are spam based on prior emails and spam reports from users, as well as on characteristics of the email itself. A few messages classified as spam are shown in figure 3.31.
- Google's Picasa and other photo-management applications can decide when a region of an image contains a human face.
- Optical character recognition software classifies small regions of scanned text into individual characters.
- Apple's Genius feature in iTunes reportedly uses classification to classify songs into potential playlists for users.

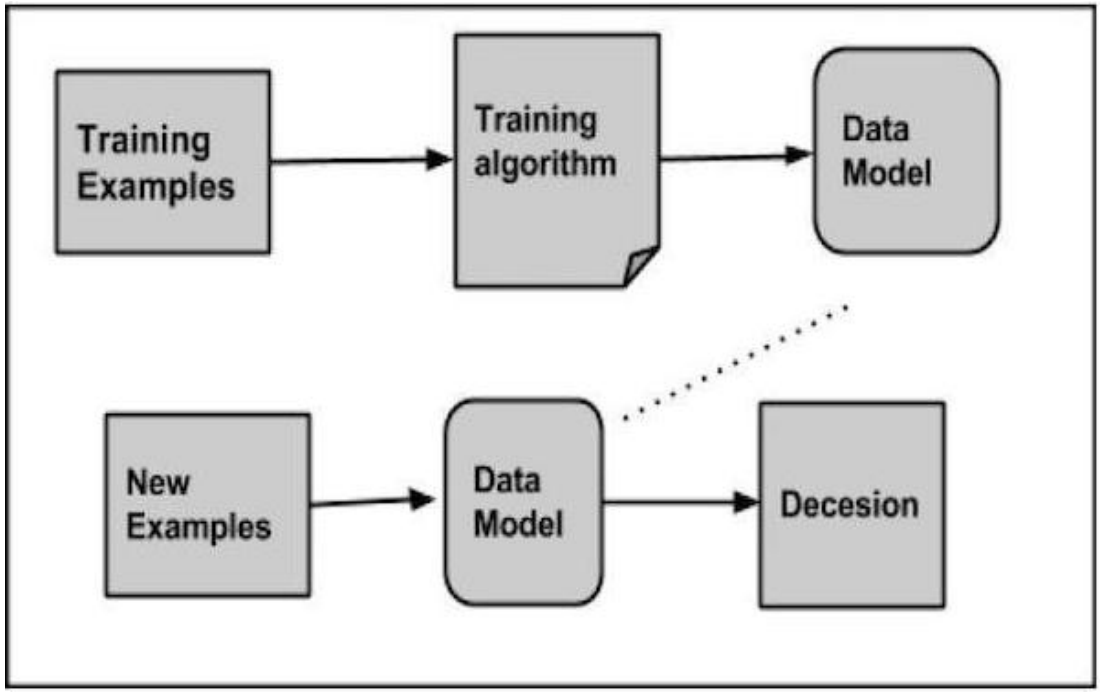
Classification helps decide whether a new input or thing matches a previously observed pattern or not, and it's often used to classify behavior or patterns as unusual.
It could be used to detect suspicious network activity or fraud. It might be used to figure out when a user's message indicates frustration or satisfaction.
Each of these techniques works best when provided with a large amount of good input data. In some cases, these techniques must not only work on large amounts of input, but must produce results quickly, and these factors make scalability a major issue.
And, as mentioned before, one of Mahout's key reasons for being is to produce implementations of these techniques that do scale up to huge input.

Figure 3.31 Spam messages as detected by Yahoo! Mail. Based on reports of email spam from users, plus other analysis, the system has learned certain attributes that usually identify spam. For example, messages mentioning "Viagra" are frequently spam—as are those with clever misspellings like "v1agra." The presence of such terms is an example of an attribute that a spam classifier can learn. [27]

How Classification Works [30]

While classifying a given set of data, the classifier system performs the following actions:

- Initially a new data model is prepared using any of the learning algorithms.
- Then the prepared data model is tested.
- Thereafter, this data model is used to evaluate the new data and to determine its class.



Applications of Classification [30]

- **Credit card fraud detection** - The Classification mechanism is used to predict credit card frauds. Using historical information of previous frauds, the classifier can predict which future transactions may turn into frauds.

- **Spam e-mails** - Depending on the characteristics of previous spam mails, the classifier determines whether a newly encountered e-mail should be sent to the spam folder.

Naive Bayes Classifier [30]

Mahout uses the Naive Bayes classifier algorithm. It uses two implementations:

- Distributed Naive Bayes classification
- Complementary Naive Bayes classification

Naive Bayes is a simple technique for constructing classifiers. It is not a single algorithm for training such classifiers, but a family of algorithms. A Bayes classifier constructs models to classify problem instances. These classifications are made using the available data.

An advantage of naive Bayes is that it only requires a small amount of training data to estimate the parameters necessary for classification.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting.

Despite its oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations.

Procedure of Classification [30]

The following steps are to be followed to implement Classification:

- Generate example data
- Create sequence files from data
- Convert sequence files to vectors
- Train the vectors
- Test the vectors

Step1: Generate Example Data
Generate or download the data to be classified. For example, you can get the 20 newsgroups example data from the following link: http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz

Create a directory for storing input data. Download the example as shown below.

```
$ mkdir classification_example
$ cd classification_example
$tar xzvf 20news-bydate.tar.gz
wget http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz
```

Step 2: Create Sequence Files
Create sequence file from the example using **seqdirectory** utility. The syntax to generate sequence is given below:

```
mahout seqdirectory -i <input file path> -o <output directory>
```

Step 3: Convert Sequence Files to Vectors

Create vector files from sequence files using **seq2parse** utility. The options of **seq2parse** utility are given below:

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName  The class name of the analyzer
--chunkSize (-chunk) chunkSize    The chunkSize in MegaBytes.
--output (-o) output              The directory pathname for o/p
--input (-i) input                Path to job input directory.
```

Step 4: Train the Vectors

Train the generated vectors using the **trainnb** utility. The options to use **trainnb** utility are given below:

```
mahout trainnb
 -i ${PATH_TO_TFIDF_VECTORS}
 -el
 -o ${PATH_TO_MODEL}/model
 -li ${PATH_TO_MODEL}/labelindex
 -ow
 -c
```

Step 5: Test the Vectors

Test the vectors using **testnb** utility. The options to use **testnb** utility are given below:

```
mahout testnb
 -i ${PATH_TO_TFIDF_TEST_VECTORS}
 -m ${PATH_TO_MODEL}/model
 -l ${PATH_TO_MODEL}/labelindex
 -ow
 -o ${PATH_TO_OUTPUT}
 -c
 -seq
```

3.2.4 Tackling large scale with Mahout and Hadoop [27]

How real is the problem of scale in machine learning algorithms? Let's consider the size of a few problems where you might deploy Mahout.

Consider that Picasa may have hosted over half a billion photos even three years ago, according to some crude estimates.[29] This implies millions of new photos per day that must be analyzed. The analysis of one photo by itself isn't a large problem, even though it's repeated millions of times. But the learning phase can require information from each of the billions of photos simultaneously—a computation on a scale that isn't feasible for a single machine.

According to a similar analysis, Google News sees about 3.5 million new news articles per day. Although this does not seem like a large amount in absolute terms, consider that these articles must be clustered, along with other recent articles, in minutes in order to become available in a timely manner.

The subset of rating data that Netflix published for the Netflix Prize contained 100 million ratings. Because this was just the data released for contest purposes, presumably the total amount of data that Netflix actually has and must process to create recommendations is many times larger!

Machine learning techniques must be deployed in contexts like these, where the amount of input is large—so large that it isn't feasible to process it all on one computer, even a powerful one. Without an implementation such as Mahout, these would be impossible tasks. This is why Mahout makes scalability a top priority. Sophisticated machine learning techniques, applied at scale, were until recently only something that large, advanced technology companies could consider using. But today computing power is cheaper than ever and more accessible via open source frameworks like Apache's Hadoop. Mahout attempts to complete the puzzle by providing quality, open source implementations capable of solving problems at this scale with Hadoop, and putting this into the hands of all technology organizations.

Some of Mahout makes use of Hadoop, which includes an open source, Java-based implementation of the MapReduce distributed computing framework popularized and used internally at Google.

MapReduce is a programming paradigm that at first sounds odd, or too simple to be powerful. The MapReduce paradigm applies to problems where the input is a set of key-value pairs. A map function turns these key-value pairs into other intermediate key value pairs. A reduce function merges in some way all values for each intermediate key to produce output. Actually, many problems can be framed as MapReduce problems, or as a series of them. The paradigm also lends itself quite well to parallelization: all of the processing is independent and so can be split across many machines.

Hadoop implements the MapReduce paradigm, which is no small feat, even given how simple MapReduce sounds. It manages storage of the input, intermediate key value pairs,

and output; this data could potentially be massive and must be available to many worker machines, not just stored locally on one. It also manages partitioning and data transfer between worker machines, as well as detection of and recovery from individual machine failures. Understanding how much work goes on behind the scenes will help prepare you for how relatively complex using Hadoop can seem. It's not just a library you add to your project. It's several components, each with libraries and (several) standalone server processes, which might be run on several machines.

Operating processes based on Hadoop isn't simple, but investing in a scalable, distributed implementation can pay dividends later: your data may quickly grow to great size, and this sort of scalable implementation is a way to future-proof your application.

CHAPTER 4

MACHINE LEARNING, STREAMS AND DATABASE ON SPARK

4.1 SPARK: REAL TIME CLUSTER COMPUTING FRAMEWORK [36]

Apache Spark is an open-source cluster computing framework for real-time processing. It is of the most successful projects in the Apache Software Foundation. Spark has clearly evolved as the market leader for Big Data processing. Today, Spark is being adopted by major players like Amazon, eBay, and Yahoo! Many organizations run Spark on clusters with thousands of nodes. We are excited to begin this exciting journey through this Spark Tutorial blog. This blog is the first blog in the upcoming Apache Spark blog series which will include Spark Streaming, Spark Interview Questions, Spark MLlib and others.

When it comes to Real Time Data Analytics, Spark stands as the go-to tool across all other solutions. Through this blog, I will introduce you to this new exciting domain of Apache Spark and we will go through a complete use case, Earthquake Detection using Spark.

4.1.1 Real Time Analytics [36]

Before we begin, let us have a look at the amount of data generated every minute by social media leaders.

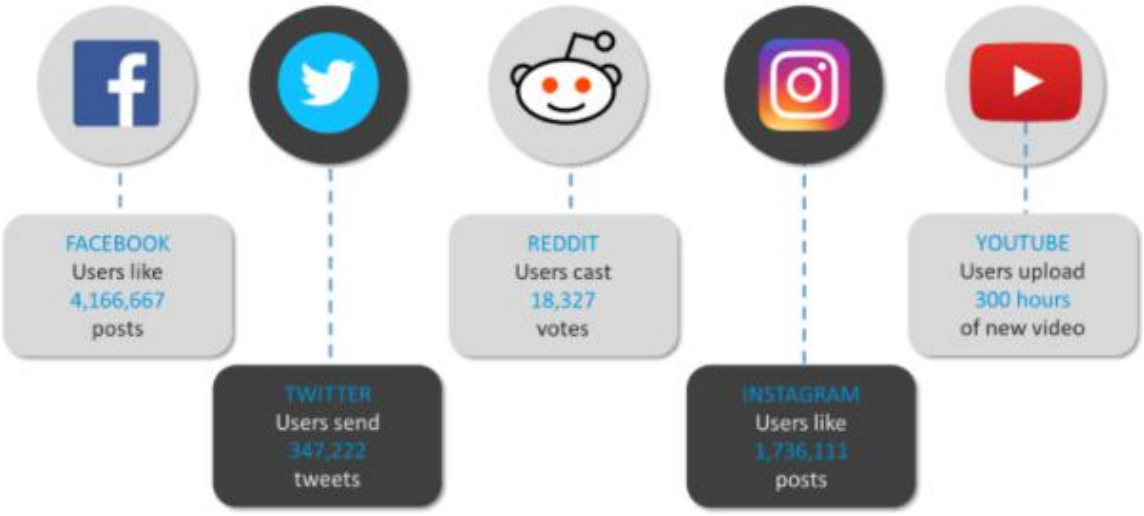


Figure 4.1 Amount of data generated every minute. [36]

As we can see, there is a colossal amount of data that the internet world necessitates to process in seconds. We will go through all the stages of handling big data in enterprises and discover the need for a *Real Time Processing Framework* called **Apache Spark**.

To begin with, let me introduce you to few domains using real-time analytics big time in today's world.

Figure 4.2 Examples of Real Time Analytics. [36]

We can see that Real Time Processing of Big Data is ingrained in every aspect of our lives. From fraud detection in banking to live surveillance systems in government, automated machines in healthcare to live prediction systems in the stock market, everything around us revolves around processing big data in near real time.

Let us look at some of these use cases of Real Time Analytics:

1. **Healthcare**: Healthcare domain uses Real Time analysis to continuously check the medical status of critical patients. Hospitals on the look out for blood and organ transplants need to stay in a real-time contact with each other during emergencies. Getting medical attention on time is a matter of life and death for patients.
2. **Government**: Government agencies perform Real Time Analysis mostly in the field of national security. Countries need to continuously keep a track of all the military and police agencies for updates regarding threats to security.
3. **Telecommunications**: Companies revolving around services in the form of calls, video chats and streaming use real-time analysis to reduce customer churn and stay

ahead of the competition. They also extract measurements of jitter and delay in mobile networks to improve customer experiences.

4. **Banking**: Banking transacts with almost all of the world's money. It becomes very important to ensure fault tolerant transactions across the whole system. Fraud detection is made possible through real-time analytics in banking.
5. **Stock Market**: Stockbrokers use real-time analytics to predict the movement of stock portfolios. Companies re-think their business model after using real-time analytics to analyze the market demand for their brand.

4.1.2 Why Spark when Hadoop is already there? [36]

The first of the many questions everyone asks when it comes to Spark is, "Why Spark when we have Hadoop already?".

To answer this, we have to look at the concept of batch and real-time processing. Hadoop is based on the concept of batch processing where the processing happens of blocks of data that have already been stored over a period of time. At the time, Hadoop broke all the expectations with the revolutionary MapReduce framework in 2005. Hadoop MapReduce is the best framework for processing data in batches.

This went on until 2014, till Spark overtook Hadoop. The USP for Spark was that it could process data in real time and was about 100 times faster than Hadoop MapReduce in batch processing large data sets.

The following figure gives a detailed explanation of the differences between processing in Spark and Hadoop.

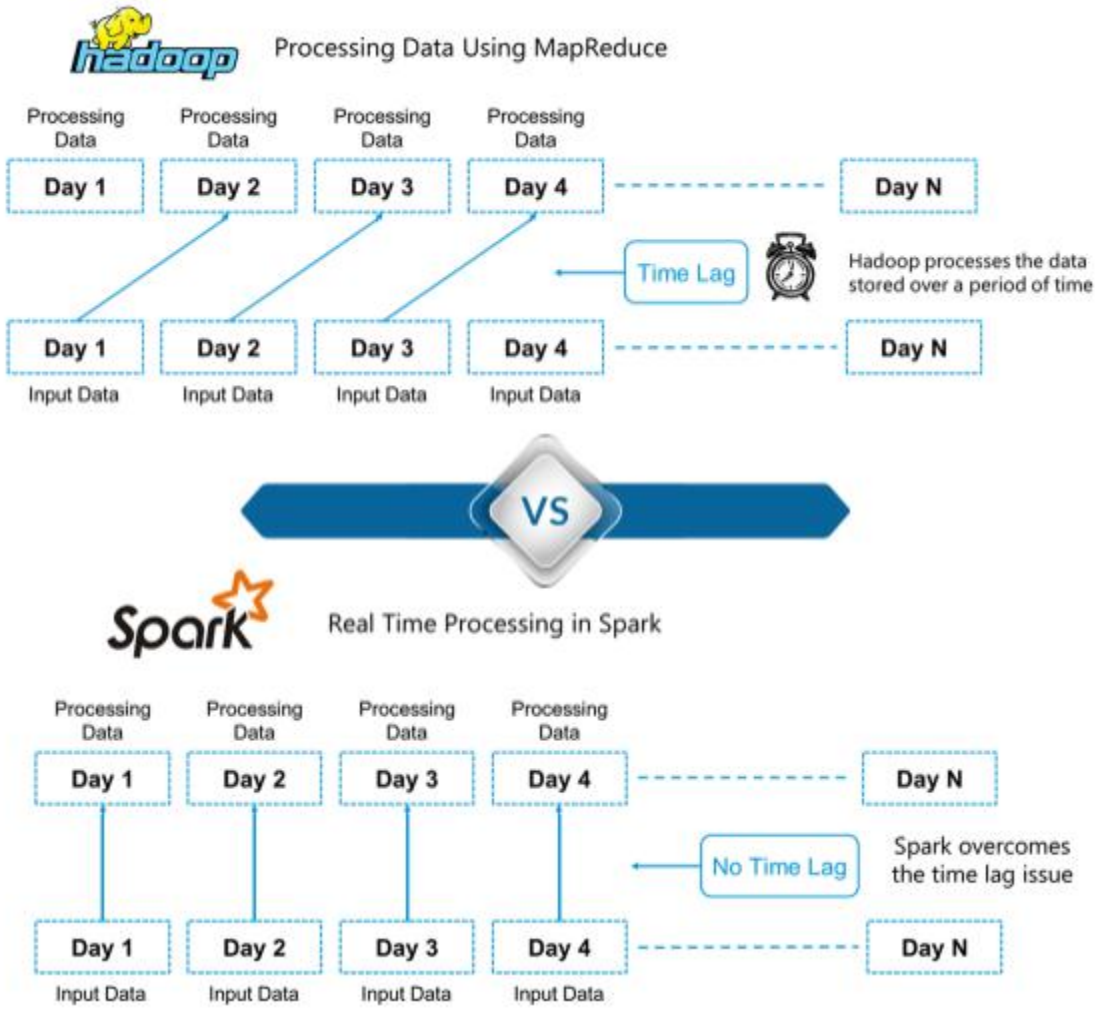


Figure 4.3 Differences between Hadoop and Spark. [36]

Here, we can draw out one of the key differentiators between Hadoop and Spark. Hadoop is based on batch processing of big data. This means that the data is stored over a period of time and is then processed using Hadoop. Whereas in Spark, processing can take place in real-time. This real-time processing power in Spark helps us to solve the use cases of Real Time Analytics we saw in the previous section. Alongside this, Spark is also able to do batch processing 100 times faster than that of Hadoop MapReduce (Processing framework in Apache Hadoop). Therefore, Apache Spark is the go-to tool for big data processing in the industry.

4.1.3 What is Apache Spark? [36]

Apache Spark is an open-source cluster computing framework for real-time processing. It has a thriving open-source community and is the most active Apache project at the

moment. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.
It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations.

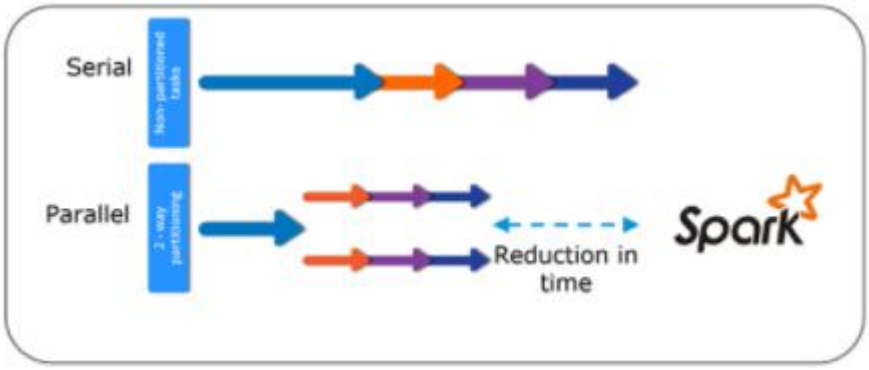


Figure 4.4 Real Time Processing in Apache Spark. [36]

4.1.4 Features of Apache Spark [36]

Spark has the following features:

Figure 4.5 Spark Features. [36]

Let us look at the features in detail:

Polyglot:
Spark provides high-level APIs in Java, Scala, Python and R. Spark code can be written in any of these four languages. It provides a shell in Scala and Python. The Scala shell can be accessed through ./bin/spark-shell and Python shell through ./bin/pyspark from the installed directory.

Speed:
Spark runs up to 100 times faster than Hadoop MapReduce for large-scale data processing. Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.

Multiple Formats:
Spark supports multiple data sources such as Parquet, JSON, Hive and Cassandra apart from the usual formats such as text files, CSV and RDBMS tables. The Data Source API provides a pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.

Lazy Evaluation:
Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For transformations, Spark adds them to a DAG (Directed Acyclic Graph) of computation and only when the driver requests some data, does this DAG actually gets executed.

Real Time Computation:
Spark's computation is real-time and has low latency because of its in-memory computation. Spark is designed for massive scalability and the Spark team has documented users of the system running production clusters with thousands of nodes and supports several computational models.

Hadoop Integration:
Apache Spark provides smooth compatibility with Hadoop. This is a boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.

Machine Learning:
Spark's MLlib is the machine learning component which is handy when it comes to big data processing. It eradicates the need to use multiple tools, one for processing and one for machine learning. Spark provides data engineers and data scientists with a powerful, unified engine that is both fast and easy to use.

4.1.5 Getting Started With Spark [36]

The first step in getting started with Spark is installation. Let us install Apache Spark 2.1.0 on Linux systems:

Installation:

1. The prerequisites for installing Spark is having Java and Scala installed.
2. Download Java in case it is not installed using below commands.

```
sudo apt-get install python-software-properties
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

3. Download the latest Scala version from Scala Lang Official page[37]. Once installed, set the scala path in ~/.bashrc file as shown below.

```
export SCALA_HOME=Path_Where_Scala_File_Is_Located
export PATH=$SCALA_HOME/bin:PATH
```

4. Download Spark 2.1.0 from the Apache Spark Downloads page[38]. You can also choose to download a previous version.
5. Extract Spark tar using below command.

```
tar -xvf spark-2.1.0-bin-hadoop2.7.tgz
```

6. Set the Spark_Path in ~/.bashrc file.

```
export SPARK_HOME=Path_Where_Spark_Is_Installed
export PATH=$PATH:$SPARK_HOME/bin
```

Before we move further, let us start up Apache Spark on our systems and get used to the main concepts of Spark like Spark Session, Data Sources, RDDs, DataFrames and other libraries.

Spark Shell:
Spark's shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively.

Spark Session:
In earlier versions of Spark, Spark Context was the entry point for Spark. For every other API, we needed to use different contexts. For streaming, we needed StreamingContext, for SQL sqlContext and for hive HiveContext. To solve this issue, SparkSession came into the picture. It is essentially a combination of SQLContext, HiveContext and future StreamingContext.

Data Sources:
The Data Source API provides a pluggable mechanism for accessing structured data though Spark SQL. Data Source API is used to read and store structured and semi-structured data into Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.

RDD:
Resilient Distributed Dataset (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Dataset:
A Dataset is a distributed collection of data. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). The Dataset API is available in Scala and Java.

DataFrames:
A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases or existing RDDs.

4.1.6 Using Spark with Hadoop [36]

The best part of Spark is its compatibility with Hadoop. As a result, this makes for a very powerful combination of technologies. Here, we will be looking at how Spark can benefit from the best of Hadoop.

Hadoop components can be used alongside Spark in the following ways:

- **HDFS**: Spark can run on top of HDFS to leverage the distributed replicated storage.

- **MapReduce**: Spark can be used along with MapReduce in the same Hadoop cluster or separately as a processing framework.
- **YARN**: Spark applications can be made to run on YARN (Hadoop NextGen).
- **Batch & Real Time Processing**: MapReduce and Spark are used together where MapReduce is used for batch processing and Spark for real-time processing.

4.1.7 Spark Components [36]

Spark components are what make Apache Spark fast and reliable. A lot of these Spark components were built to resolve the issues that cropped up while using Hadoop MapReduce. Apache Spark has the following components:

1. Spark Core
2. Spark Streaming
3. Spark SQL
4. GraphX
5. MLlib (Machine Learning)

Spark Core

Spark Core is the base engine for large-scale parallel and distributed data processing. The core is the distributed execution engine and the Java, Scala, and Python APIs offer a platform for distributed ETL application development. Further, additional libraries which are built atop the core allow diverse workloads for streaming, SQL, and machine learning. It is responsible for:

1. Memory management and fault recovery
2. Scheduling, distributing and monitoring jobs on a cluster
3. Interacting with storage systems

Spark Streaming

Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams. The fundamental stream unit is DStream which is basically a series of RDDs (Resilient Distributed Datasets) to process the real-time data.

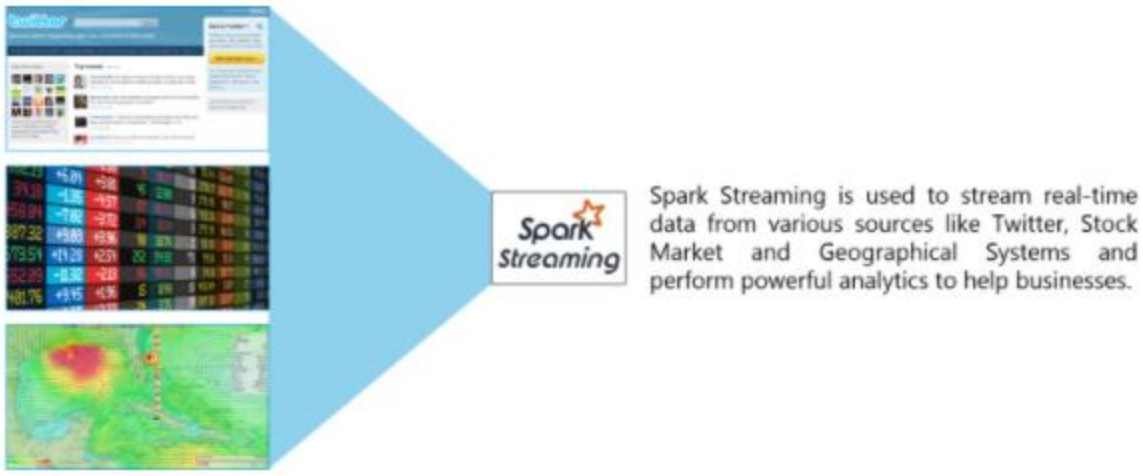


Figure 4.6 Spark Streaming. [36]

Spark SQL

Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.

Spark SQL integrates relational processing with Spark's functional programming. Further, it provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool.

The following are the four libraries of Spark SQL.

1. Data Source API
2. DataFrame API
3. Interpreter & Optimizer
4. SQL Service

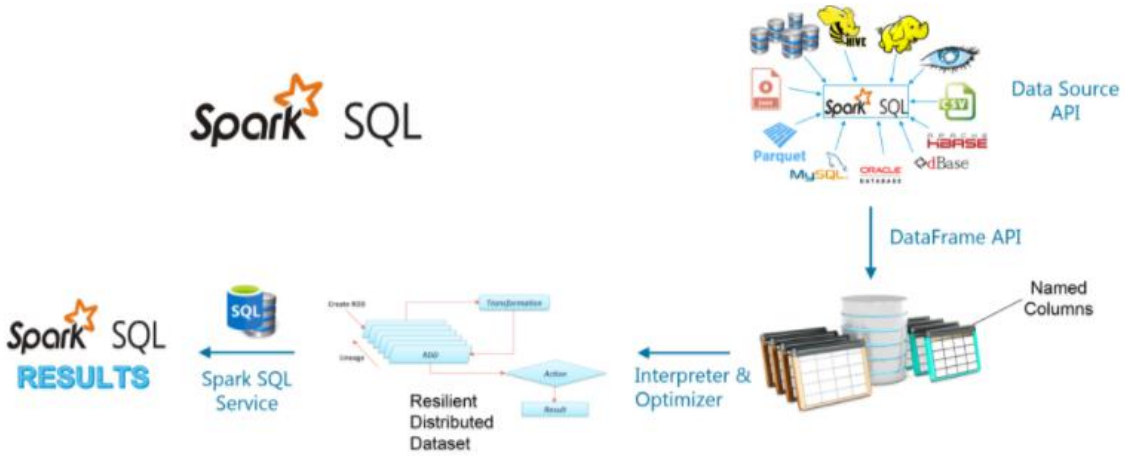


Figure 4.7 Spark SQL process using all the four libraries in sequence. [36]

GraphX

GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph.

The property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. Here, the parallel edges allow multiple relationships between the same vertices. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge.

To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

MlLib (Machine Learning)

MLlib stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.

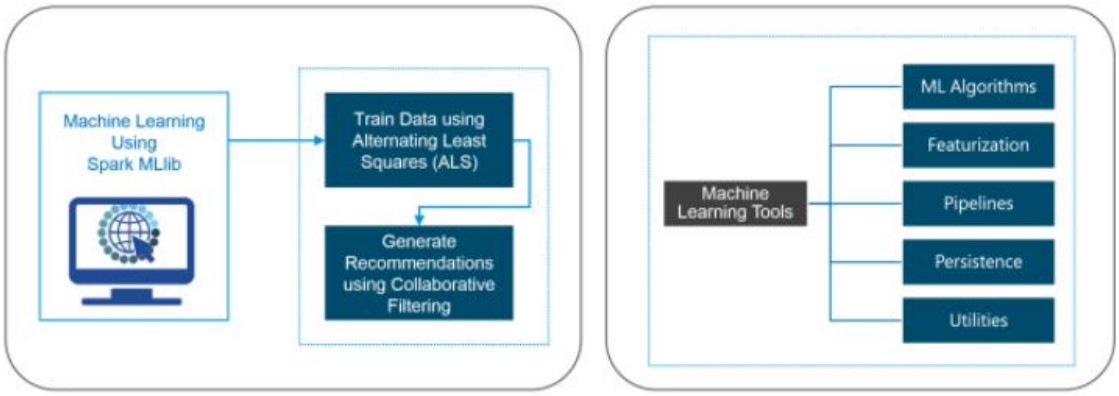


Figure 4.8 Machine Learning Flow Diagram / Machine Learning Tools. [36]

4.1.8 Earthquake Detection using Spark [36]

Now that we have understood the core concepts of Spark, let us solve a real-life problem using Apache Spark. This will help give us the confidence to work on any Spark projects in the future.

Problem Statement: *To design a Real Time Earthquake Detection Model to send lifesaving alerts, which should improve its machine learning to provide near real-time computation results.*

Use Case – Requirements:

1. Process data in real-time
2. Handle input from multiple sources
3. Easy to use system
4. Bulk transmission of alerts

We will use Apache Spark which is the perfect tool for our requirements.

Use Case – Dataset:

EARTHQUAKE ROC DATASET																		
	S Wave							P Wave										
Classification Index	First Activity	Time Taken	Acceleration	Building Strength	Velocity	Sa	Sd	First Activity	Time Taken	Acceleration	Building Strength	Velocity	Sa	Sd	Total Weight	Sum * ROC	ROC	AVG * ROC
0	3	11	14	19	39	42	55	64	67	73	75	76	80	83	701	618.168	0.881837	623.2843
0	3	6	17	27	35	40	57	63	69	73	74	76	81	103	724	638.4503	0.881837	623.2843
0	4	6	15	21	35	40	57	63	67	73	74	77	80	83	695	612.877	0.881837	623.2843
0	5	6	15	22	36	41	47	66	67	72	74	76	80	83	690	608.4678	0.881837	623.2843
0	2	6	16	22	36	40	54	63	67	73	75	76	80	83	693	611.1133	0.881837	623.2843
0	2	6	14	20	37	41	47	64	67	73	74	76	82	83	686	604.9405	0.881837	623.2843
0	1	6	14	22	36	42	49	64	67	72	74	77	80	83	687	605.8223	0.881837	623.2843
0	1	6	17	19	39	42	53	64	67	73	74	76	80	83	694	611.9952	0.881837	623.2843
0	2	6	18	20	37	42	48	64	71	73	74	76	81	83	695	612.877	0.881837	623.2843
1	5	11	15	32	39	40	52	63	67	73	74	76	78	83	708	624.3409	0.881837	623.2843
0	5	16	30	35	41	64	67	73	74	76	80	83			644	567.9033	0.881837	623.2843
0	5	6	15	20	37	40	50	63	67	73	75	76	80	83	690	608.4678	0.881837	623.2843
0	5	7	16	29	39	40	48	63	67	73	74	76	78	83	698	615.5225	0.881837	623.2843
0	1	11	18	20	37	42	59	62	71	72	74	76	80	83	706	622.5772	0.881837	623.2843
1	5	18	19	39	40	63	67	73	74	76	80	83			637	561.7304	0.881837	623.2843

Figure 4.9 Use Case – Earthquake Dataset. [36]

You can download the complete dataset from [39]

Before moving ahead, there is one concept we have to learn that we will be using in our Earthquake Detection System and it is called Receiver Operating Characteristic (ROC). An ROC curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. We will use the dataset to obtain an ROC value using Machine Learning in Apache Spark.

Use Case – Flow Diagram:

The following illustration clearly explains all the steps involved in our Earthquake Detection System.

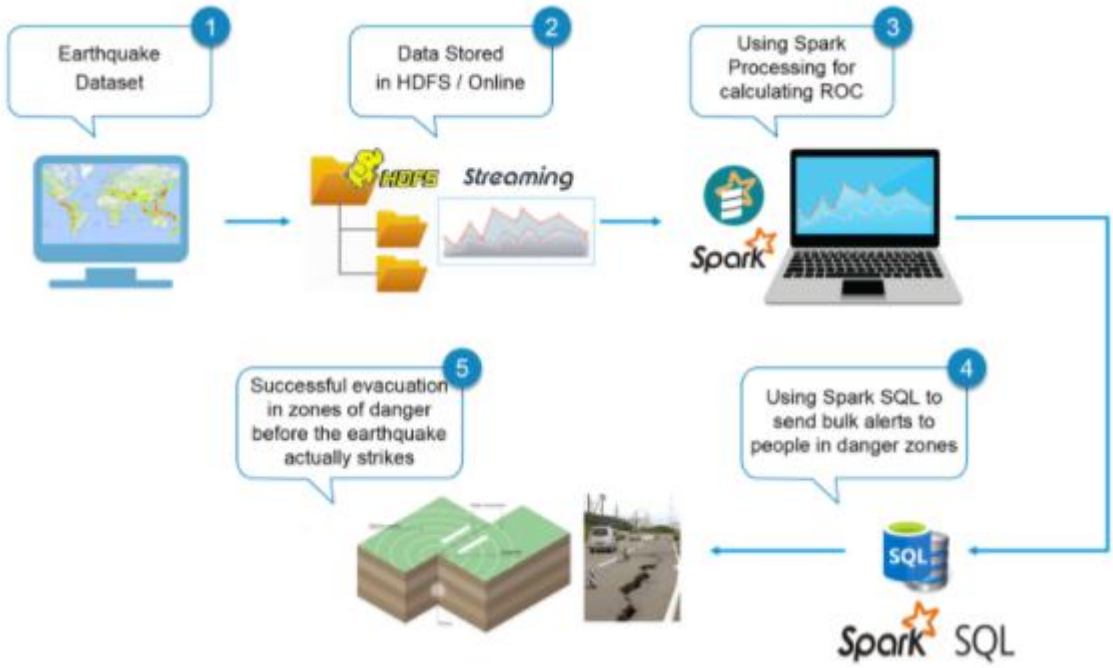


Figure 4.10 Flow diagram of Earthquake Detection using Apache Spark. [36]

Use Case – Spark Implementation:

Moving ahead, now let us implement our project using Eclipse IDE for Spark.

Find the Pseudo Code below:

```
//Importing the necessary classes
import org.apache.spark._
...
//Creating an Object earthquake
object earthquake {
 def main(args: Array[String]) {

//Creating a Spark Configuration and Spark Context
val sparkConf = new SparkConf().setAppName("earthquake").setMaster("local[2]")
val sc = new SparkContext(sparkConf)

//Loading the Earthquake ROC Dataset file as a LibSVM file
val data = MLUtils.loadLibSVMFile(sc, *Path to the Earthquake File* )

//Training the data for Machine Learning
val splits = data.randomSplit( *Splitting 60% to 40%* , seed = 11L)
val training = splits(0).cache()
val test = splits(1)

//Creating a model of the trained data
val numIterations = 100
val model = *Creating SVM Model with SGD* (  *Training Data* , *Number of Iterations* )

//Using map transformation of model RDD
val scoreAndLabels = *Map the model to predict features*

//Using Binary Classification Metrics on scoreAndLabels
val metrics = * Use Binary Classification Metrics on scoreAndLabels *(scoreAndLabels)
val auROC = metrics. *Get the area under the ROC Curve*()

//Displaying the area under Receiver Operating Characteristic
println("Area under ROC = " + auROC)
 }
}
```

The full source code of Earthquake Detection using Apache Spark is available in [40]

From our Spark program, we obtain the ROC value to be 0.088137. We will be transforming this value to get the area under the ROC curve.

Use Case – Visualizing Results:

We will plot the ROC curve and compare it with the specific earthquake points. Where ever the earthquake points exceed the ROC curve, such points are treated as major earthquakes. As per our algorithm to calculate the Area under the ROC curve, we can assume that these major earthquakes are above 6.0 magnitude on the Richter scale.

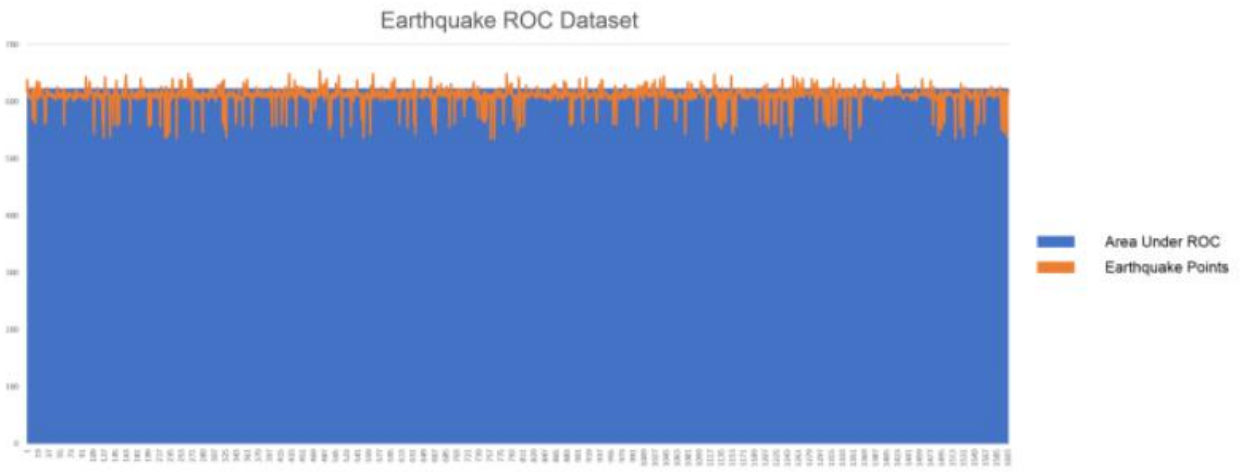


Figure 4.11 Earthquake ROC Curve. [36]

The above image shows the Earthquake line in orange. The area in blue is the ROC curve that we have obtained from our Spark program. Let us zoom into the curve to get a better picture.

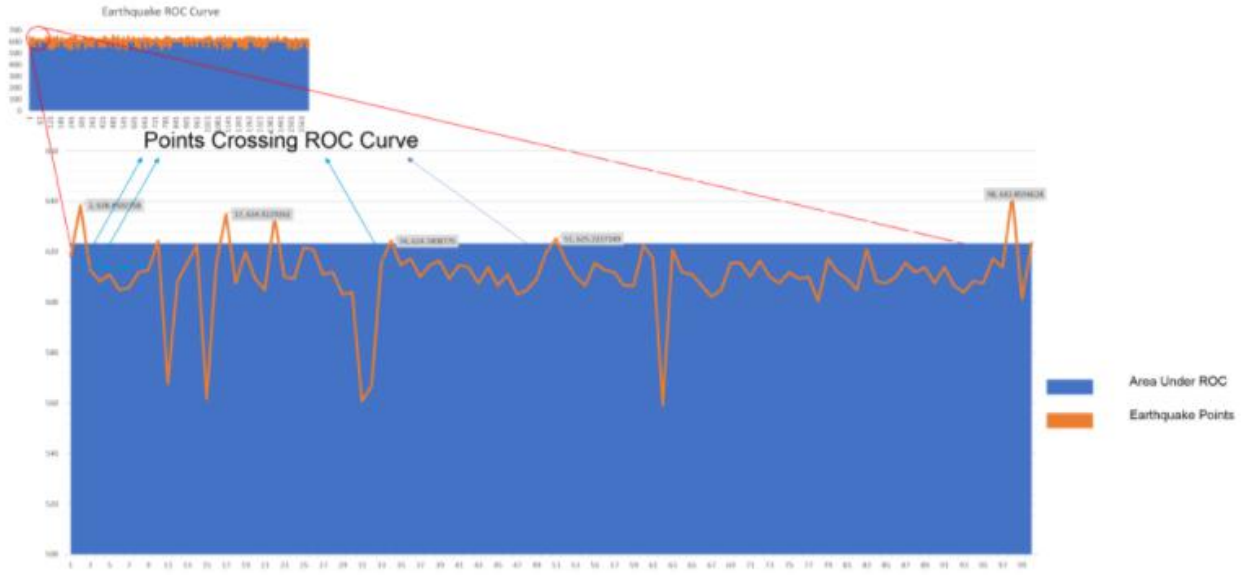


Figure 4.12 Visualizing Earthquake Points. [36]

We have plotted the earthquake curve against the ROC curve. At points where the orange curve is above the blue region, we have predicted the earthquakes to be major, i.e., with magnitude greater than 6.0. Thus armed with this knowledge, we could use Spark SQL and query an existing Hive table to retrieve email addresses and send people personalized warning emails. Thus we have used technology once more to save human life from trouble and make everyone's life better.

4.2 SENTIMENT ANALYSIS USING APACHE SPARK [36]

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming can be used to stream live data and processing can happen in real time. Spark Streaming's ever-growing user base consists of household names like Uber, Netflix and Pinterest.

When it comes to Real Time Data Analytics, Spark Streaming provides a single platform to ingest data for fast and live processing in Apache Spark. Through this blog, I will introduce you to this new exciting domain of Spark Streaming and we will go through a complete use case, Twitter Sentiment Analysis using Spark Streaming.

4.2.1 What is Streaming? [36]

Data Streaming is a technique for transferring data so that it can be processed as a steady and continuous stream. Streaming technologies are becoming increasingly important with the growth of the Internet.

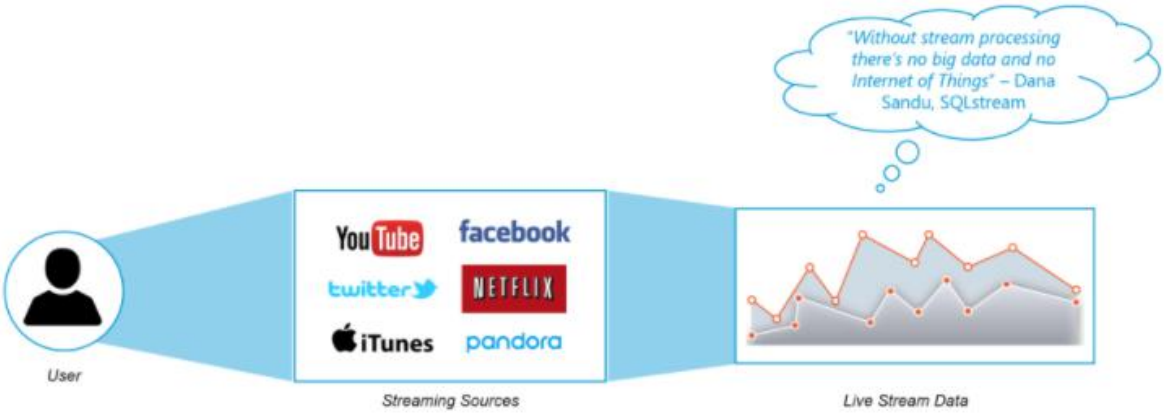


Figure 4.13 What is Streaming? [36]

4.2.2 Why Spark Streaming? [36]

We can use Spark Streaming to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

4.2.3 Spark Streaming Overview [36]

Spark Streaming is used for processing real-time streaming data. It is a useful addition to the core Spark API. Spark Streaming enables high-throughput and fault-tolerant stream processing of live data streams.

Figure 4.14 Streams in Spark Streaming [36]

The fundamental stream unit is DStream which is basically a series of RDDs to process the real-time data.

4.2.4 Spark Streaming Features [36]

1. **Scaling**: Spark Streaming can easily scale to hundreds of nodes.
2. **Speed**: It achieves low latency.
3. **Fault Tolerance**: Spark has the ability to efficiently recover from failures.
4. **Integration**: Spark integrates with batch and real-time processing.
5. **Business Analysis**: Spark Streaming is used to track the behavior of customers which can be used in business analysis.

4.2.5 Spark Streaming Workflow [36]

Spark Streaming workflow has four high-level stages. The first is to stream data from various sources. These sources can be streaming data sources like Akka, Kafka, Flume, AWS or Parquet for real-time streaming. The second type of sources includes HBase, MySQL, PostgreSQL, Elastic Search, Mongo DB and Cassandra for static/batch streaming. Once this happens, Spark can be used to perform Machine Learning on the data through its MLlib API. Further, Spark SQL is used to perform further operations on this data. Finally, the streaming output can be stored into various data storage systems like HBase, Cassandra, MemSQL, Kafka, Elastic Search, HDFS and local file system.

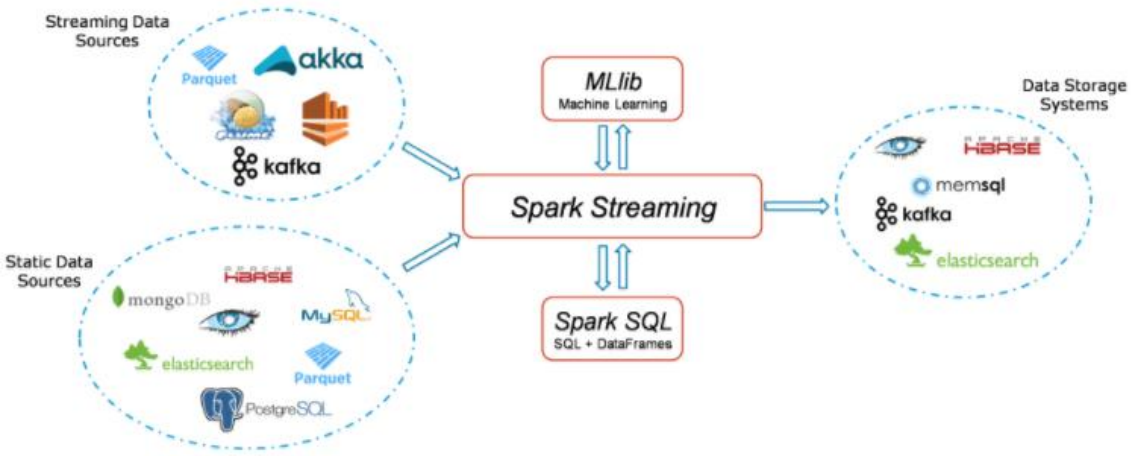


Figure 4.15 Overview Of Spark Streaming [36]

4.2.6 Spark Streaming Fundamentals [36]

1. Streaming Context
2. DStream
3. Caching
4. Accumulators, Broadcast Variables and Checkpoints

***Streaming Context* [36]**

Streaming Context consumes a stream of data in Spark. It registers an Input DStream to produce a Receiver object. It is the main entry point for Spark functionality. Spark provides a number of default implementations of sources like Twitter, Akka Actor and ZeroMQ that are accessible from the context.

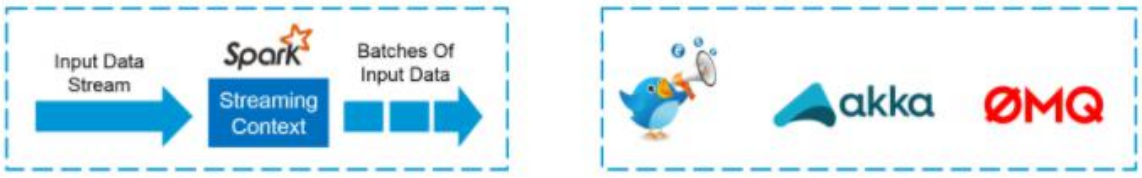


Figure 4.16 Spark Streaming Context / Default Implementation Sources [36]

A StreamingContext object can be created from a SparkContext object. A SparkContext represents the connection to a Spark cluster and can be used to create RDDs, accumulators and broadcast variables on that cluster.

```
import org.apache.spark._
import org.apache.spark.streaming._
var ssc = new StreamingContext(sc,Seconds(1))
```

DStream [36]

Discretized Stream (DStream) is the basic abstraction provided by Spark Streaming. It is a continuous stream of data. It is received from a data source or a processed data stream generated by transforming the input stream.

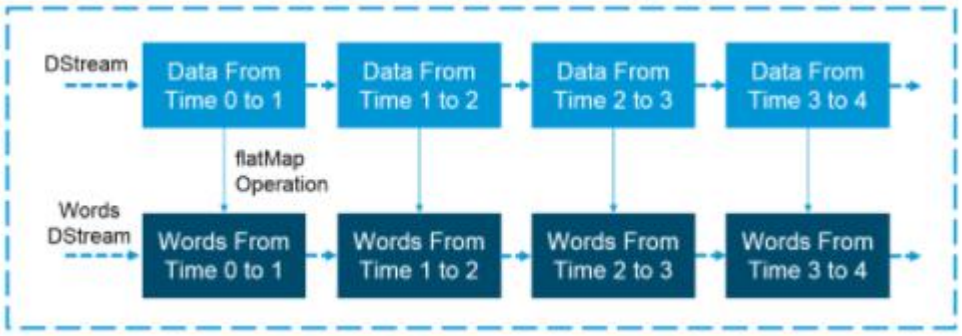


Figure 4.17 Extracting words from an Input DStream [36]

Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval.

Input DStreams: Input DStreams are DStreams representing the stream of input data received from streaming sources.

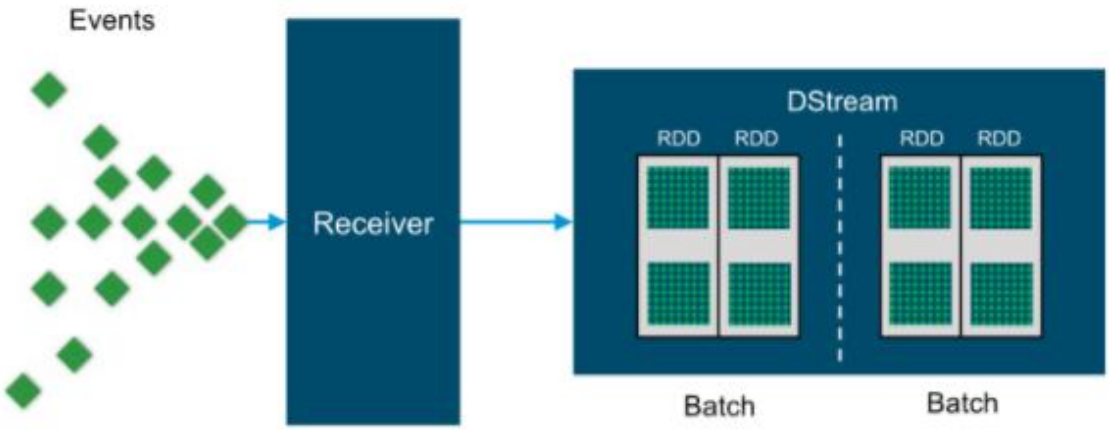


Figure 4.18 The Receiver sends data onto the Input DStream where each Batch contains RDDs [36]

Every input DStream is associated with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.

Transformations on DStreams:

Any operation applied on a DStream translates to operations on the underlying RDDs. Transformations allow the data from the input DStream to be modified similar to RDDs. DStreams support many of the transformations available on normal Spark RDDs.

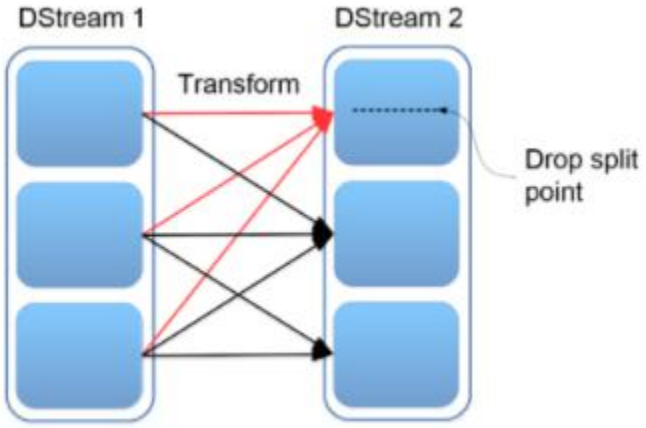


Figure 4.19 DStream Transformations [36]

The following are some of the popular transformations on DStreams:

map(*func*)	map(*func*) returns a new DStream by passing each element of the source DStream through a function *func*.
flatMap(*func*)	flatMap(*func*) is similar to map(*func*) but each input item can be mapped to 0 or more output items and returns a new DStream by passing each source element through a function *func*.
filter(*func*)	filter(*func*) returns a new DStream by selecting only the records of the source DStream on which *func* returns true.
reduce(*func*)	reduce(*func*) returns a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function *func*.
groupBy(*func*)	groupBy(*func*) returns the new RDD which basically is made up with a key and corresponding list of items of that group.

Output DStreams:

Output operations allow DStream's data to be pushed out to external systems like databases or file systems. Output operations trigger the actual execution of all the DStream transformations.

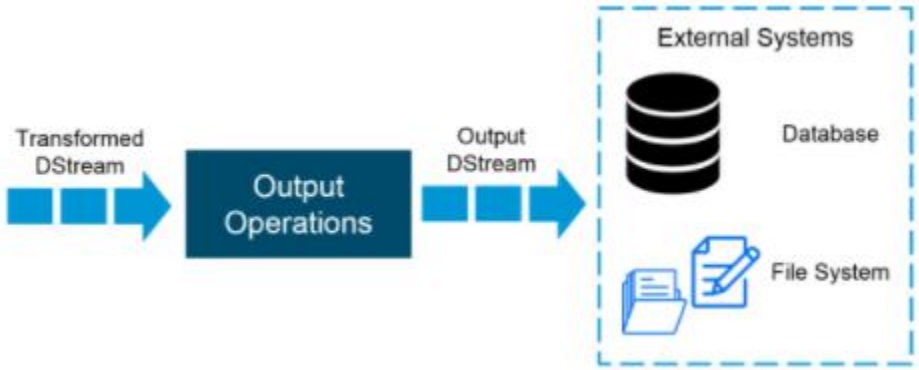


Figure 4.20 Output Operations on DStreams [36]

Caching [36]

DStreams allow developers to cache/ persist the stream's data in memory. This is useful if the data in the DStream will be computed multiple times. This can be done using the persist() method on a DStream.

Figure 4.21 Caching into 2 Nodes [36]

For input streams that receive data over the network (such as Kafka, Flume, Sockets, etc.), the default persistence level is set to replicate the data to two nodes for fault-tolerance.

Accumulators, Broadcast Variables and Checkpoints

Accumulators: Accumulators are variables that are only added through an associative and commutative operation. They are used to implement counters or sums. Tracking accumulators in the UI can be useful for understanding the progress of running stages.

Spark natively supports numeric accumulators. We can create named or unnamed accumulators.

Broadcast Variables: Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used to give every node a copy of a large input dataset in an efficient manner. Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

Checkpoints: Checkpoints are similar to checkpoints in gaming. They make it run 24/7 and make it resilient to failures unrelated to the application logic.

Figure 4.22 Features of Checkpoints [36]

4.2.7 Use Case – Twitter Sentiment Analysis [36]

Now that we have understood the core concepts of Spark Streaming, let us solve a real-life problem using Spark Streaming.

Problem Statement: *To design a Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing.*

Applications of Sentiment Analysis:

1. Predict the success of a movie
2. Predict political campaign success
3. Decide whether to invest in a certain company
4. Targeted advertising
5. Review products and services

Spark Streaming Implementation:

Find the Pseudo Code below:

```
//Import the necessary packages into the Spark Program
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext._
...
import java.io.File

object twitterSentiment {

def main(args: Array[String]) {
if (args.length < 4) {
System.err.println("Usage: TwitterPopularTags <consumer key> <consumer secret> "
    + "<access token> <access token secret> [<filters>]")
System.exit(1)
}

StreamingExamples.setStreamingLogLevels()
//Passing our Twitter keys and tokens as arguments for authorization
val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) =
    args.take(4)
val filters = args.takeRight(args.length - 4)

// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
...
System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)

val sparkConf = new SparkConf().setAppName("twitterSentiment").setMaster("local[2]")
val ssc = new Streaming Context
val stream = TwitterUtils.createStream(ssc, None, filters)

//Input DStream transformation using flatMap
val tags = stream.flatMap { status => Get Text From The Hashtags }

//RDD transformation using sortBy and then map function
tags.countByValue()
.foreachRDD { rdd =>
val now = Get current time of each Tweet
rdd
.sortBy(_._2)
.map(x => (x, now))
//Saving our output at ~/twitter/ directory
.saveAsTextFile(s"~/twitter/$now")
```

```
}

//DStream transformation using filter and map functions
val tweets = stream.filter {t =>
val tags = t. Split On Spaces .filter(_.startsWith("#")). Convert To Lower Case
tags.exists { x => true }
}

val data = tweets.map { status =>
val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
val tagss = status.getHashtagEntities.map(_.getText.toLowerCase)
(status.getText, sentiment.toString, tagss.toString())
}

data.print()
//Saving our output at ~/ with filenames starting like twitters
data.saveAsTextFiles("~/twitters","20000")

ssc.start()
ssc.awaitTermination()
 }
}
```

The full source code of Twitter Sentiment Analysis using Spark Streaming is available in [41]

Results:

The following are the results that are displayed in the Eclipse IDE while running the Twitter Sentiment Streaming program.

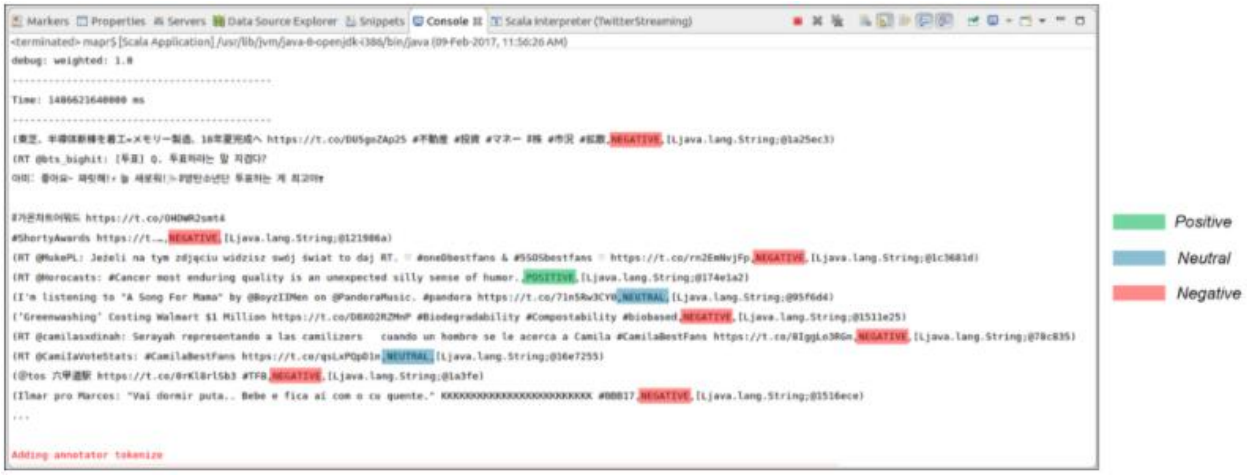


Figure 4.23 Sentiment Analysis Output in Eclipse IDE [36]

As we can see in the screenshot, all the tweets are categorized into Positive, Neutral and Negative according to the sentiment of the contents of the tweets.

The output of the Sentiments of the Tweets is stored into folders and files according to the time they were created. This output can be stored on the local file system or HDFS as necessary. The output directory looks like this:

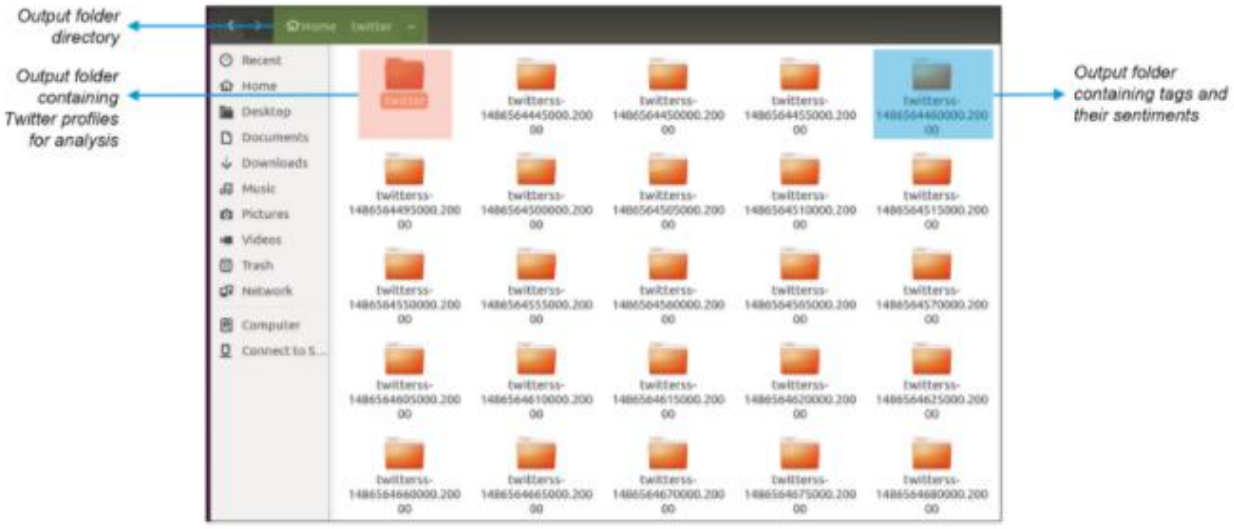


Figure 4.24 Output folders inside our 'twitter' project folder [36]

Here, inside the twitter directory, we can find the usernames of the Twitter users along with the timestamp for every tweet as shown below:

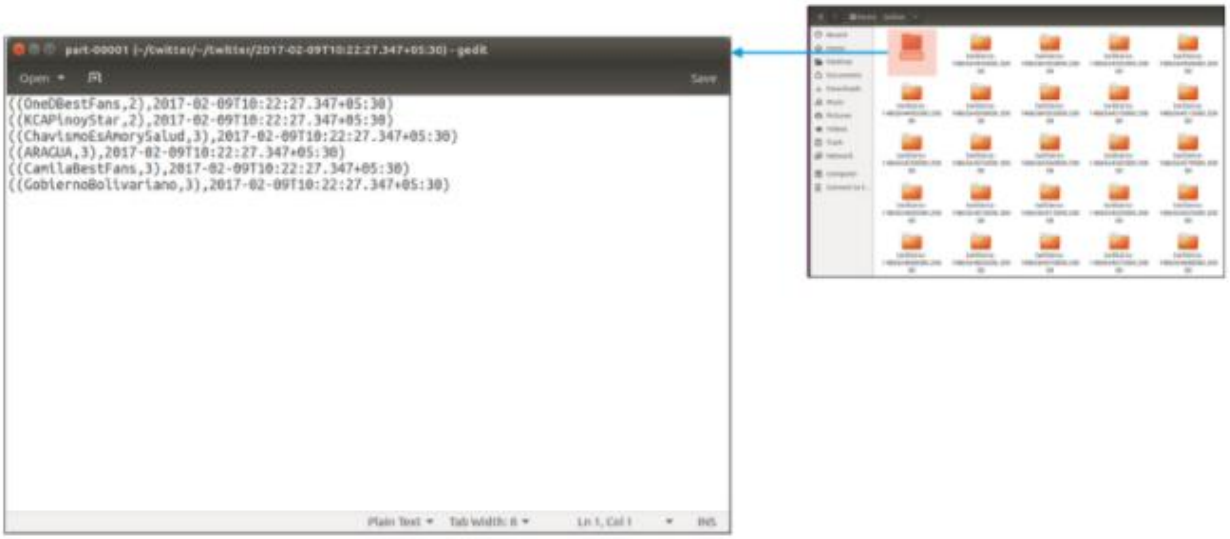


Figure 4.25 Output file containing Twitter usernames with timestamp [36]

Now that we have got the Twitter usernames and timestamp, let us look at the Sentiments and tweets stored in the main directory. Here, every tweet is followed by the sentiment emotion. This Sentiment that is stored is further used for analyzing a vast multitude of insights by companies.

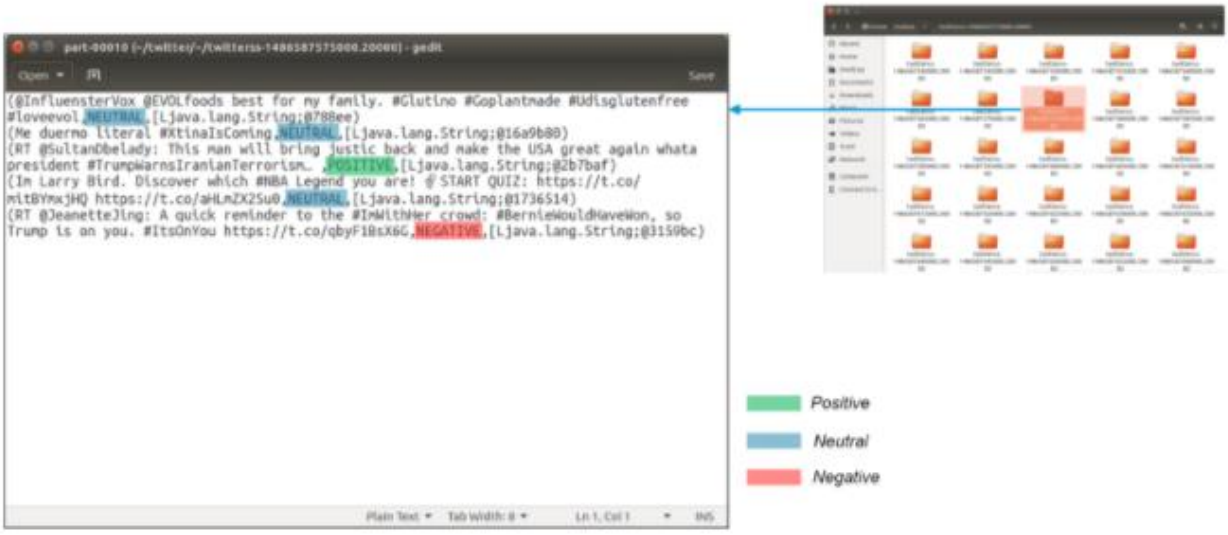


Figure 4.26 Output file containing tweets with sentiments [36]

Tweaking Code:

Now, let us modify our code a little to get sentiments for specific hashtags (topics). Currently, Donald Trump, the President of the United States is trending across news channels and online social media. Let us look at the sentiments associated with the keyword ‘Trump‘.

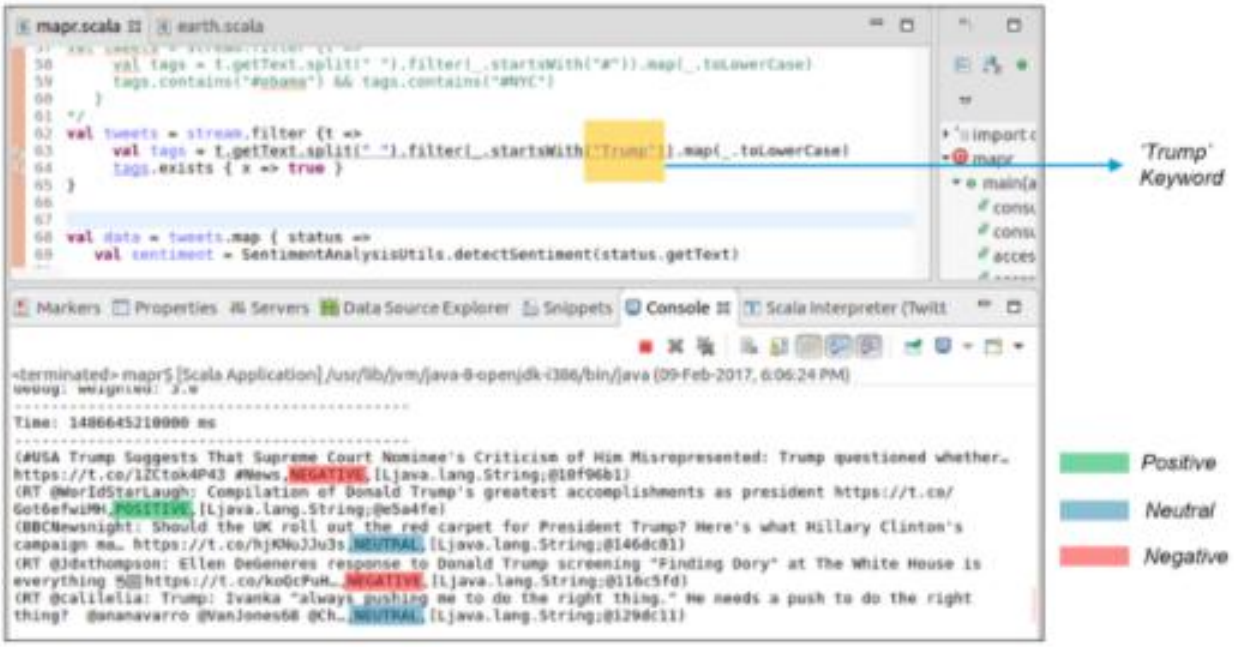


Figure 4.27 Performing Sentiment Analysis on Tweets with 'Trump' Keyword [36]

Moving Ahead:

As we have seen from our Sentiment Analysis demonstration, we can extract sentiments of particular topics just like we did for 'Trump'. Similarly, Sentiment Analytics can be used in crisis management, service adjusting and target marketing by companies around the world.

Companies using Spark Streaming for Sentiment Analysis have applied the same approach to achieve the following:

1. Enhancing the customer experience
2. Gaining competitive advantage
3. Gaining Business Intelligence
4. Revitalizing a losing brand

4.3 SPARK MLLIB – MACHINE LEARNING LIBRARY OF APACHE SPARK [36]

Spark MLlib is Apache Spark's Machine Learning component. One of the major attractions of Spark is the ability to scale computation massively, and that is exactly what you need for machine learning algorithms. But the limitation is that all machine learning

algorithms cannot be effectively parallelized. Each algorithm has its own challenges for parallelization, whether it is task parallelism or data parallelism.

Having said that, Spark is becoming the de-facto platform for building machine learning algorithms and applications. The developers working on the Spark MLlib are implementing more and more machine algorithms in a scalable and concise manner in the Spark framework. Through this blog, we will learn the concepts of Machine Learning, Spark MLlib, its utilities, algorithms and a complete use case of Movie Recommendation System.

4.3.1 What is Machine Learning? [36]

Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.

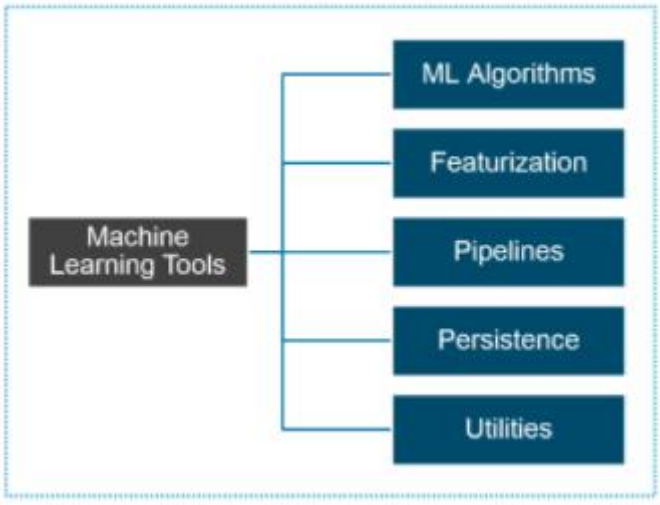


Figure 4.28 Machine Learning tools [36]

Machine learning is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to a prediction which in commercial use is known as predictive analytics.

There are three categories of Machine learning tasks:

1. **Supervised Learning**: Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.
2. **Unsupervised Learning**: Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.
3. **Reinforcement Learning**: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space. This concept is called reinforcement learning.

4.3.2 Spark MLlib Overview [36]

Spark MLlib is used to perform machine learning in Apache Spark. MLlib consists popular algorithms and utilities.

MLlib Overview:

- spark.mllib contains the original API built on top of RDDs. It is currently in maintenance mode.
- spark.ml provides higher level API built on top of DataFrames for constructing ML pipelines. spark.ml is the primary Machine Learning API for Spark at the moment.

4.3.3 Spark MLlib Tools [36]

Spark MLlib provides the following tools:

- **ML Algorithms**: ML Algorithms form the core of MLlib. These include common learning algorithms such as classification, regression, clustering and collaborative filtering.
- **Featurization**: Featurization includes feature extraction, transformation, dimensionality reduction and selection.
- **Pipelines**: Pipelines provide tools for constructing, evaluating and tuning ML Pipelines.

- **Persistence**: Persistence helps in saving and loading algorithms, models and Pipelines.
- **Utilities**: Utilities for linear algebra, statistics and data handling.

4.3.4 MLlib Algorithms [36]

The popular algorithms and utilities in Spark MLlib are:

1. Basic Statistics
2. Regression
3. Classification
4. Recommendation System
5. Clustering
6. Dimensionality Reduction
7. Feature Extraction
8. Optimization

Let us look at some of these in detail.

Basic Statistics
Basic Statistics includes the most basic of machine learning techniques. These include:

1. **Summary Statistics**: Examples include mean, variance, count, max, min and numNonZeros.
2. **Correlations**: Spearman and Pearson are some ways to find correlation.
3. **Stratified Sampling**: These include sampleBykey and sampleByKeyExact.
4. **Hypothesis Testing**: Pearson's chi-squared test is an example of hypothesis testing.
5. **Random Data Generation**: RandomRDDs, Normal and Poisson are used to generate random data.

Regression
Regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables when the focus is on the relationship between a dependent variable and one or more independent variables. More specifically, regression analysis helps one understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to

understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer causal relationships between the independent and dependent variables.

Classification

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. It is an example of pattern recognition.

Here, an example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.).

Recommendation System

A recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item. Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general.

Recommender systems typically produce a list of recommendations in one of two ways – through collaborative and content-based filtering or the personality-based approach.

- **Collaborative Filtering** approaches building a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in.
- **Content-Based Filtering** approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties.

Further, these approaches are often combined as Hybrid Recommender Systems.

Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). So, it is the main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine

learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression and computer graphics.

Dimensionality Reduction
Dimensionality Reduction is the process of reducing the number of random variables under consideration, via obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

- **Feature Selection**: Feature selection finds a subset of the original variables (also called features or attributes).
- **Feature Extraction**: This transforms the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in Principal Component Analysis(PCA), but many nonlinear dimensionality reduction techniques also exist.

Feature Extraction
Feature Extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. This is related to dimensionality reduction.

Optimization
Optimization is the selection of the best element (with regard to some criterion) from some set of available alternatives.

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.

4.3.5 Use Case – Movie Recommendation System [36]

Problem Statement: To build a Movie Recommendation System which recommends movies based on a user's preferences using Apache Spark.

Our Requirements:
So, let us assess the requirements to build our movie recommendation system:

1. Process huge amount of data
2. Input from multiple sources
3. Easy to use
4. Fast processing

As we can assess our requirements, we need the best Big Data tool to process large data in short time. Therefore, Apache Spark is the perfect tool to implement our Movie Recommendation System.

Let us now look at the Flow Diagram for our system.

Figure 4.29 Flow Diagram for the system [36]

As we can see, the following uses Streaming from Spark Streaming. We can stream in real time or read data from Hadoop HDFS.

Getting Dataset:

For our Movie Recommendation System, we can get user ratings from many popular websites like IMDB, Rotten Tomatoes and Times Movie Ratings. This dataset is available in many formats such as CSV files, text files and databases. We can either stream the data live from the websites or download and store them in our local file system or HDFS.

Dataset:

The below figure shows how we can collect dataset from popular websites.

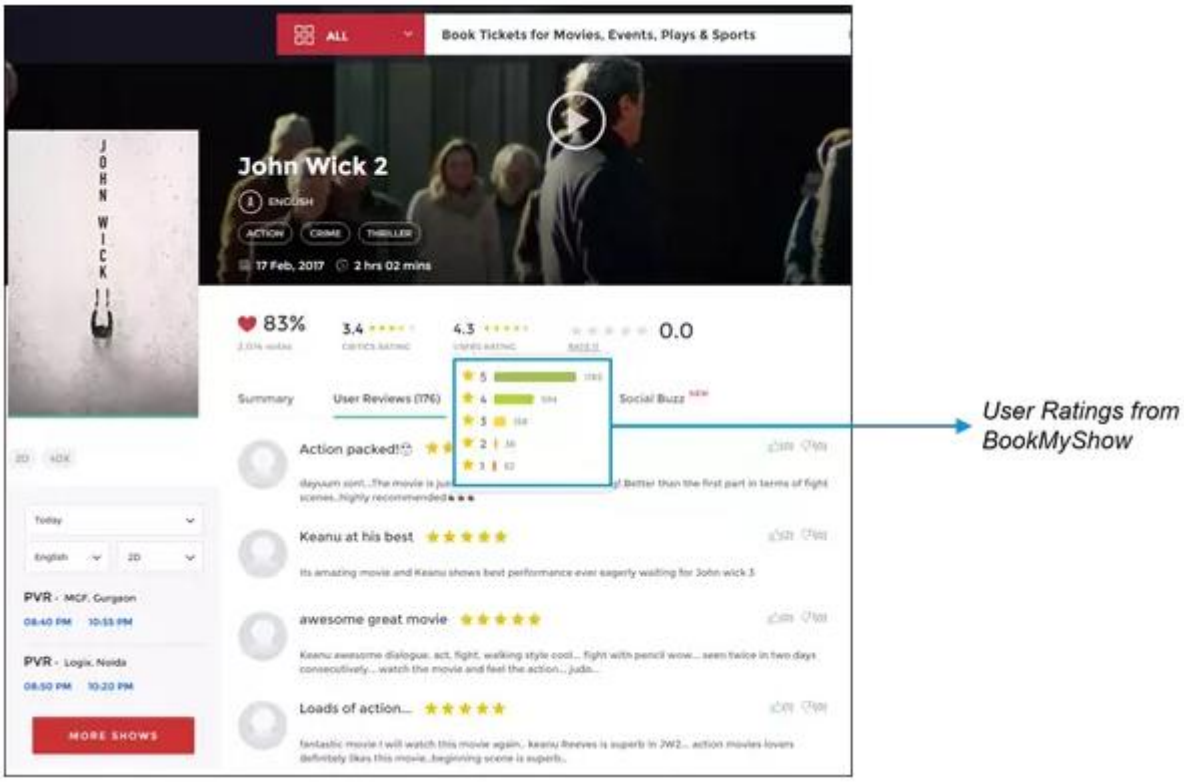
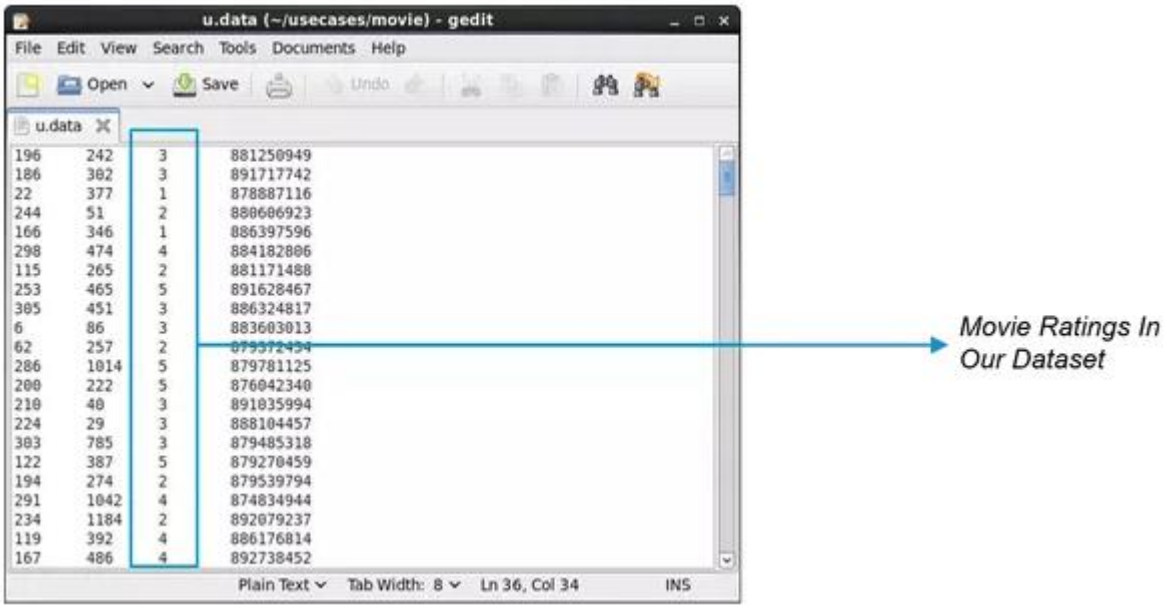


Figure 4.30 How to collect dataset from popular websites [36]

Once we stream the data into Spark, it looks somewhat like this.



Machine Learning:

The whole recommendation system is based on Machine Learning algorithm Alternating Least Squares. Here, ALS is a type of regression analysis where regression is used to draw a line amidst the data points in such a way so that the sum of the squares of the distance from each data point is minimized. Thus, this line is then used to predict the values of the function where it meets the value of the independent variable.

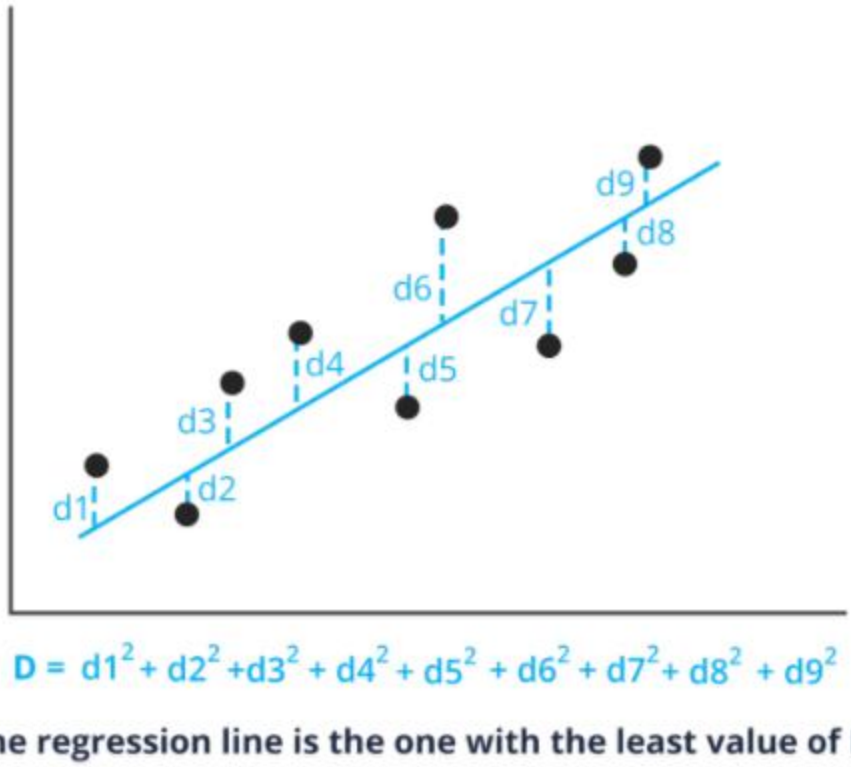


Figure 4.31 Machine Learning Algorithm – Regression Alternating Least Squares [36]

The blue line in the diagram is the best-fit regression line. For this line, the value of the dimension D is minimum. All other red lines will always be farther from the dataset as a whole.

Spark MLlib Implementation:

1. We will use Collaborative Filtering(CF) to predict the ratings for users for particular movies based on their ratings for other movies.
2. We then collaborate this with other users' rating for that particular movie.
3. To get the following results from our Machine Learning, we need to use Spark SQL's DataFrame, Dataset and SQL Service.

Here is the pseudo code for our program:

```
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating
import org.apache.spark.SparkConf
//Import other necessary packages

object Movie {
def main(args: Array[String]) {

val conf = new SparkConf().setAppName("Movie").setMaster("local[2]")
val sc = new SparkContext(conf)
val rawData = sc.textFile(" *Read Data from Movie CSV file* ")

//rawData.first()
val rawRatings = rawData.map( *Split rawData on tab delimiter* )
val ratings = rawRatings.map { *Map case array of User, Movie and Rating* }

//Training the data
val model = ALS.train(ratings, 50, 5, 0.01)
model.userFeatures
model.userFeatures.count
model.productFeatures.count
val predictedRating = *Predict for User 789 for movie 123*
val userId = *User 789*
val K = 10
val topKRecs = model.recommendProducts( *Recommend for User for the particular
value of K* )
println(topKRecs.mkString("\n"))
val movies = sc.textFile(" *Read Movie List Data* ")
val titles = movies.map(line => line.split("\\|").take(2)).map(array =>
(array(0).toInt,array(1))).collectAsMap()
val titlesRDD = movies.map(line => line.split("\\|").take(2)).map(array =>
(array(0).toInt,array(1))).cache()
titles(123)
val moviesForUser = ratings.*Search for User 789*
val sqlContext= *Create SQL Context*
val moviesRecommended = sqlContext.*Make a DataFrame of recommended movies*
moviesRecommended.registerTempTable("moviesRecommendedTable")
sqlContext.sql("Select count(*) from moviesRecommendedTable").foreach(println)
moviesForUser. *Sort the ratings for User 789* .map( *Map the rating to movie title* ).
*Print the rating*
val results = moviesForUser.sortBy(-_.rating).take(30).map(rating =>
(titles(rating.product), rating.rating))
 }
}
```

The full source code of Movie Recommendation System using Spark MLlib is available in [42].

Once we generate predictions, we can use Spark SQL to store the results into an RDBMS system. Further, this can be displayed on a web application.

Results:

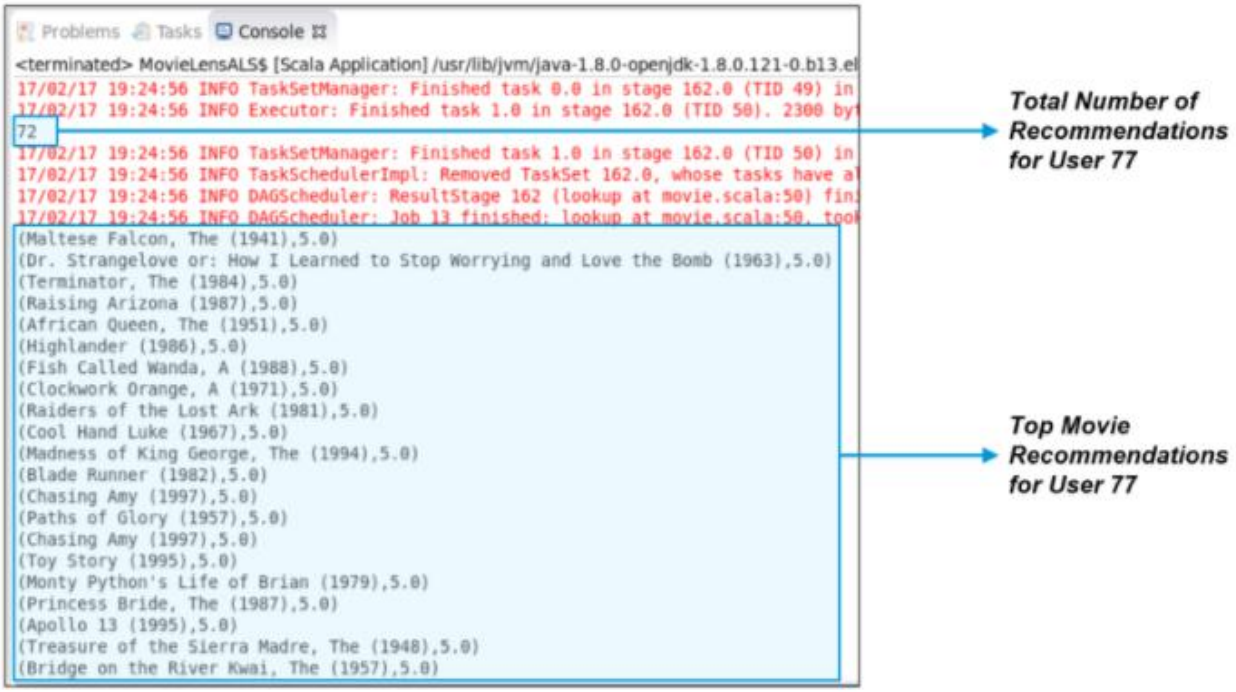


Figure 4.32 Movies recommended for User 77 [36]

4.4 SPARK SQL TUTORIAL – UNDERSTANDING SPARK SQL WITH EXAMPLES [36]

Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language.

For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing. Through this blog, I will introduce you to this new exciting domain of Spark SQL and

together we will equip ourselves to lead our organization to leverage the benefits of relational processing and call complex analytics libraries in Spark.

4.4.1 Why Spark SQL Came Into Picture? [36]

Spark SQL originated as Apache Hive to run on top of Spark and is now integrated with the Spark stack. Apache Hive had certain limitations as mentioned below. Spark SQL was built to overcome these drawbacks and replace Apache Hive.

4.4.2 Limitations with Hive: [36]

- Hive launches MapReduce jobs internally for executing the ad-hoc queries. MapReduce lags in the performance when it comes to the analysis of medium sized datasets (10 to 200 GB).
- Hive has no resume capability. This means that if the processing dies in the middle of a workflow, you cannot resume from where it got stuck.
- Hive cannot drop encrypted databases in cascade when trash is enabled and leads to an execution error. To overcome this, users have to use Purge option to skip trash instead of drop.

These drawbacks gave way to the birth of Spark SQL.

4.4.3 Spark SQL Overview [36]

Spark SQL integrates relational processing with Spark's functional programming. It provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool.

Let us explore, what Spark SQL has to offer. Spark SQL blurs the line between RDD and relational table. It offers much tighter integration between relational and procedural processing, through declarative DataFrame APIs which integrates with Spark code. It also provides higher optimization. DataFrame API and Datasets API are the ways to interact with Spark SQL.

With Spark SQL, Apache Spark is accessible to more users and improves optimization for the current ones. Spark SQL provides DataFrame APIs which perform relational operations on both external data sources and Spark's built-in distributed collections. It

introduces extensible optimizer called Catalyst as it helps in supporting a wide range of data sources and algorithms in Big-data.

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Microsoft, Mac OS). It is easy to run locally on one machine — all you need is to have java installed on your system PATH, or the JAVA_HOME environment variable pointing to a Java installation.

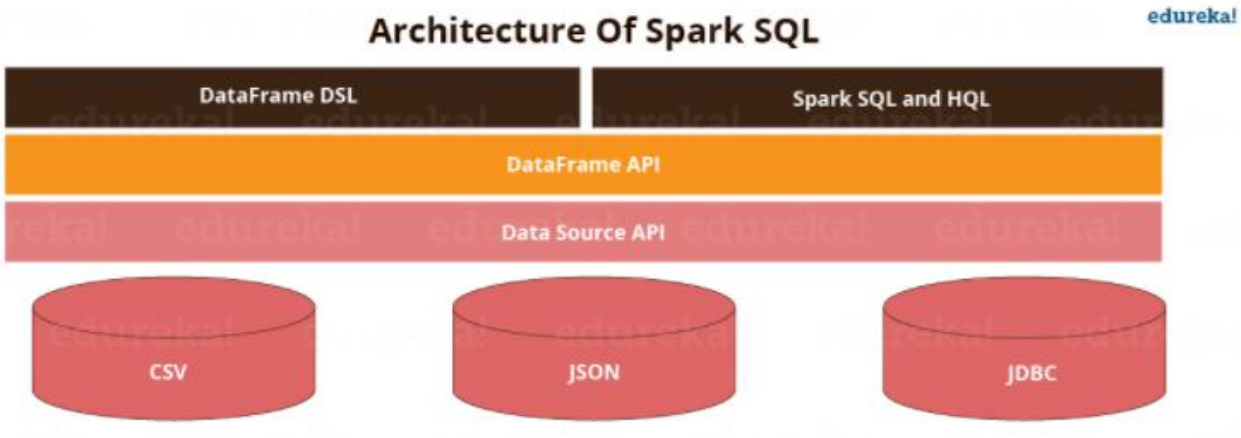


Figure 4.33 Architecture of Spark SQL. [36]

4.4.4 Spark SQL Libraries [36]

Spark SQL has the following four libraries which are used to interact with relational and procedural processing:

1. **Data Source API (Application Programming Interface):**

This is a universal API for loading and storing structured data.

- It has built in support for Hive, Avro, JSON, JDBC, Parquet, etc.
- Supports third party integration through Spark packages
- Support for smart sources.

2. **DataFrame API:**

A DataFrame is a distributed collection of data organized into named column. It is equivalent to a relational table in SQL used for storing data into tables.

- It is a Data Abstraction and Domain Specific Language (DSL) applicable on structure and semi structured data.
- DataFrame API is distributed collection of data in the form of named column and row.
- It is lazily evaluated like Apache Spark Transformations and can be accessed through SQL Context and Hive Context.
- It processes the data in the size of Kilobytes to Petabytes on a single-node cluster to multi-node clusters.
- Supports different data formats (Avro, CSV, Elastic Search and Cassandra) and storage systems (HDFS, HIVE Tables, MySQL, etc.).
- Can be easily integrated with all Big Data tools and frameworks via Spark-Core.
- Provides API for Python, Java, Scala, and R Programming.

3. SQL Interpreter And Optimizer:

SQL Interpreter and Optimizer is based on functional programming constructed in Scala.

- It is the newest and most technically evolved component of SparkSQL.
- It provides a general framework for transforming trees, which is used to perform analysis/evaluation, optimization, planning, and run time code spawning.
- This supports cost based optimization (run time and resource utilization is termed as cost) and rule based optimization, making queries run much faster than their RDD (Resilient Distributed Dataset) counterparts.

e.g. Catalyst is a modular library which is made as a rule based system. Each rule in framework focuses on the distinct optimization.

4. SQL Service:

SQL Service is the entry point for working along structured data in Spark. It allows the creation of DataFrame objects as well as the execution of SQL queries.

4.4.5 Features Of Spark SQL [36]

The following are the features of Spark SQL:

1. Integration With Spark

Spark SQL queries are integrated with Spark programs. Spark SQL allows us to query structured data inside Spark programs, using SQL or a DataFrame API which can be used in Java, Scala, Python and R. To run streaming computation, developers simply write a

batch computation against the DataFrame / Dataset API, and Spark automatically increments the computation to run it in a streaming fashion. This powerful design means that developers don't have to manually manage state, failures, or keeping the application in sync with batch jobs. Instead, the streaming job always gives the same answer as a batch job on the same data.

2. Uniform Data Access

DataFrames and SQL support a common way to access a variety of data sources, like Hive, Avro, Parquet, ORC, JSON, and JDBC. This joins the data across these sources. This is very helpful to accommodate all the existing users into Spark SQL.

3. Hive Compatibility

Spark SQL runs unmodified Hive queries on current data. It rewrites the Hive front-end and meta store, allowing full compatibility with current Hive data, queries, and UDFs.

4. Standard Connectivity

Connection is through JDBC or ODBC. JDBC and ODBC are the industry norms for connectivity for business intelligence tools.

5. Performance And Scalability

Spark SQL incorporates a cost-based optimizer, code generation and columnar storage to make queries agile alongside computing thousands of nodes using the Spark engine, which provides full mid-query fault tolerance. The interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimization. Spark SQL can directly read from multiple sources (files, HDFS, JSON/Parquet files, existing RDDs, Hive, etc.). It ensures fast execution of existing Hive queries.

The image below depicts the performance of Spark SQL when compared to Hadoop. Spark SQL executes upto 100x times faster than Hadoop.

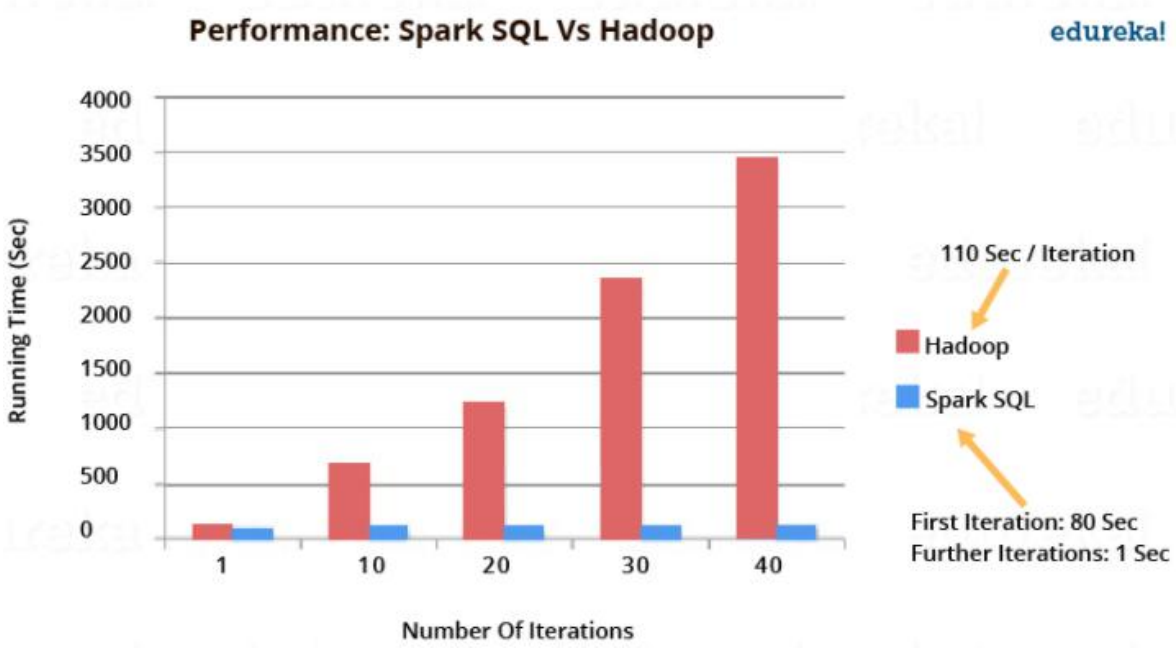


Figure 4.34 Runtime of Spark SQL vs Hadoop. Spark SQL is faster
Source: Cloudera Apache Spark Blog. [36]

6. User Defined Functions

Spark SQL has language integrated User-Defined Functions (UDFs). UDF is a feature of Spark SQL to define new Column-based functions that extend the vocabulary of Spark SQL's DSL for transforming Datasets. UDFs are black boxes in their execution.
The example below defines a UDF to convert a given text to upper case.

Code explanation:

1. Creating a dataset "hello world"
2. Defining a function 'upper' which converts a string into upper case.
3. We now import the 'udf' package into Spark.
4. Defining our UDF, 'upperUDF' and importing our function 'upper'.
5. Displaying the results of our User Defined Function in a new column 'upper'.

```
1  val dataset = Seq((0, "hello"),(1, "world")).toDF("id","text")
2  val upper: String => String =_.toUpperCase
3  import org.apache.spark.sql.functions.udf
4  val upperUDF = udf(upper)
5  dataset.withColumn("upper", upperUDF('text')).show
```

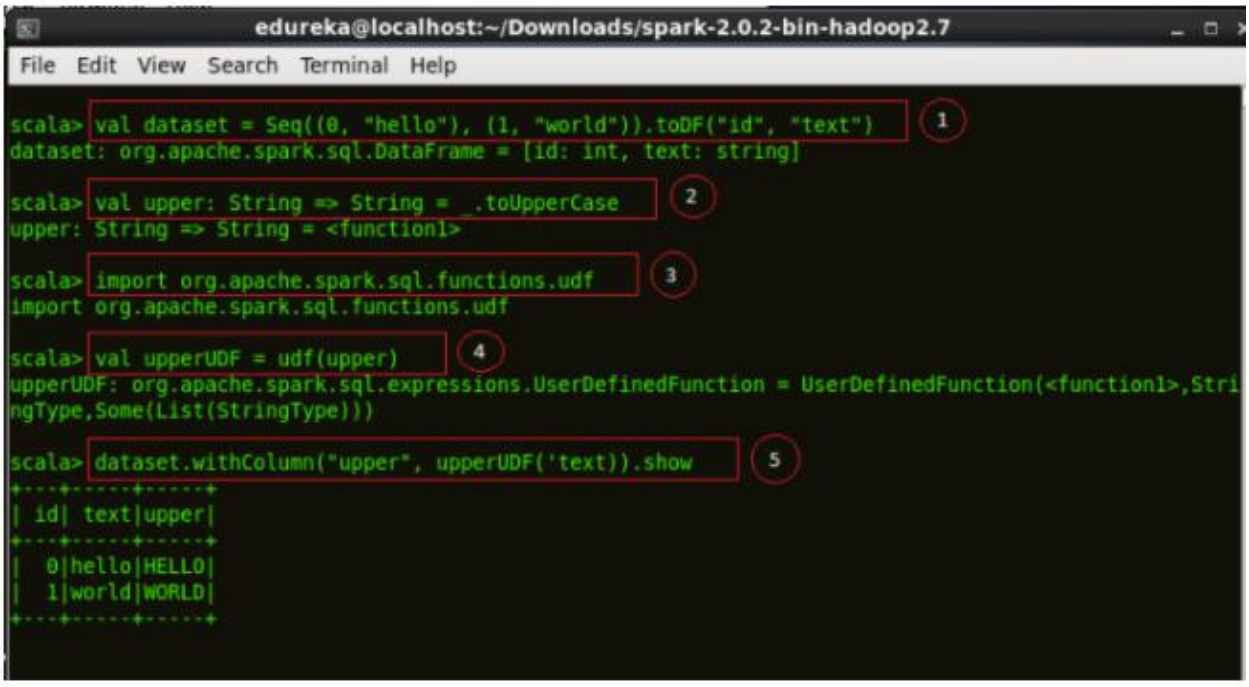


Figure 4.35 Demonstration of a User Defined Function, upperUDF. [36]

Code explanation:

1. We now register our function as 'myUpper'
2. Cataloging our UDF among the other functions.

```
1  spark.udf.register("myUpper", (input:String) => input.toUpperCase)
2  spark.catalog.listFunctions.filter('name like "%upper%").show(false)
```

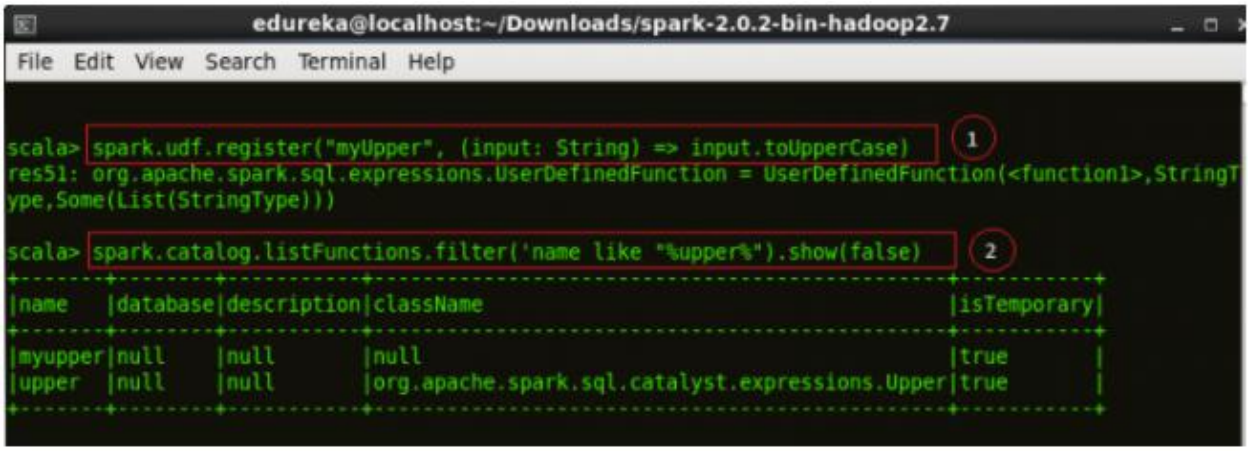


Figure 4.36 Results of the User Defined Function, upperUDF. [36]

4.4.6 Querying Using Spark SQL [36]

We will now start querying using Spark SQL. Note that the actual SQL queries are similar to the ones used in popular SQL clients.

Starting the Spark Shell. Go to the Spark directory and execute ./bin/spark-shell in the terminal to being the Spark Shell.

For the querying examples shown here, we will be using two files, 'employee.txt' and 'employee.json'. The images below show the content of both the files. Both these files are stored at 'examples/src/main/scala/org/apache/spark/examples/sql/SparkSQLExample.scala' inside the folder containing the Spark installation (~/Downloads/spark-2.0.2-bin-hadoop2.7). So, all of you who are executing the queries, place them in this directory or set the path to your files in the lines of code below.

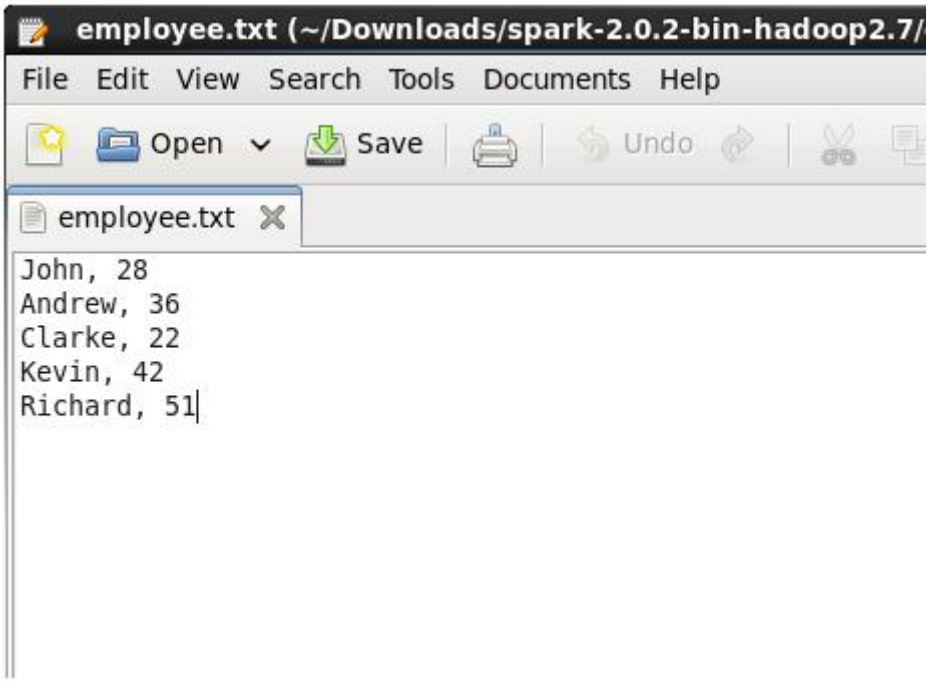


Figure 4.37 Contents of employee.txt. [36]

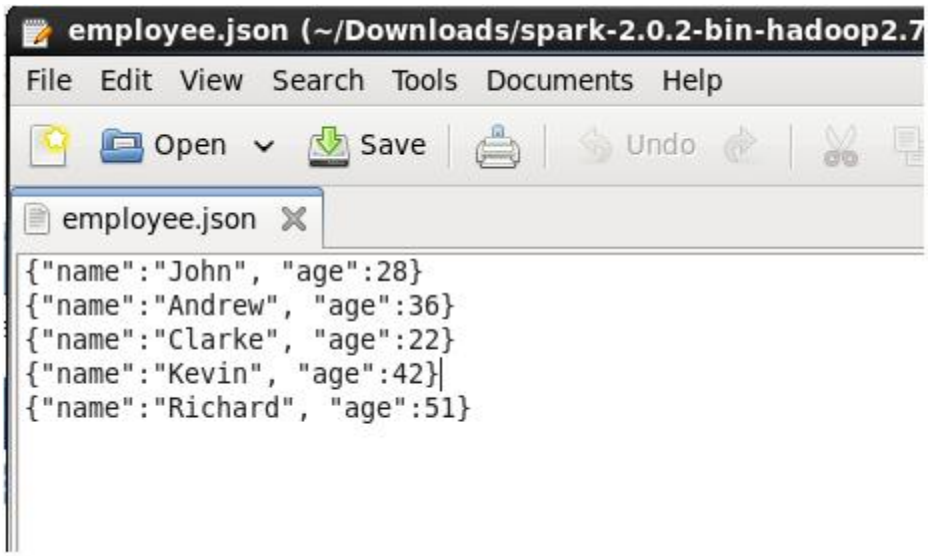


Figure 4.38 Contents of employee.json. [36]

Code explanation:

1. We first import a Spark Session into Apache Spark.
2. Creating a Spark Session 'spark' using the 'builder()' function.
3. Importing the Implicts class into our 'spark' Session.
4. We now create a DataFrame 'df' and import data from the 'employee.json' file.
5. Displaying the DataFrame 'df'. The result is a table of 5 rows of ages and names from our 'employee.json' file.

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().appName("Spark SQL basic
example").config("spark.some.config.option", "some-value").getOrCreate()
import spark.implicits._
val df = spark.read.json("examples/src/main/resources/employee.json")
df.show()
```

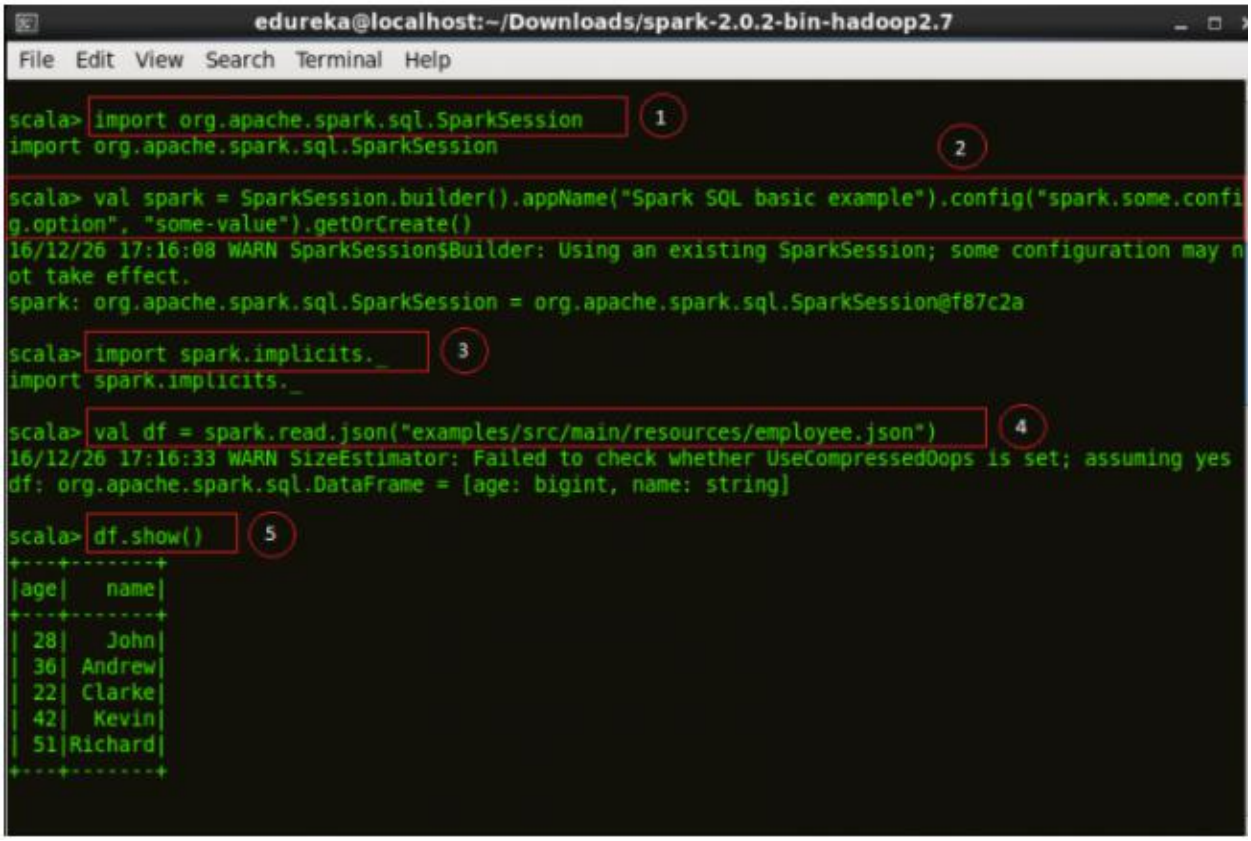


Figure 4.39 Starting a Spark Session and displaying DataFrame of employee.json. [36]

Code explanation:

1. Importing the Implicts class into our ‘spark’ Session.
2. Printing the schema of our ‘df’ DataFrame.
3. Displaying the names of all our records from ‘df’ DataFrame.

```
1  import spark.implicits._
2  df.printSchema()
3  df.select("name").show()
```

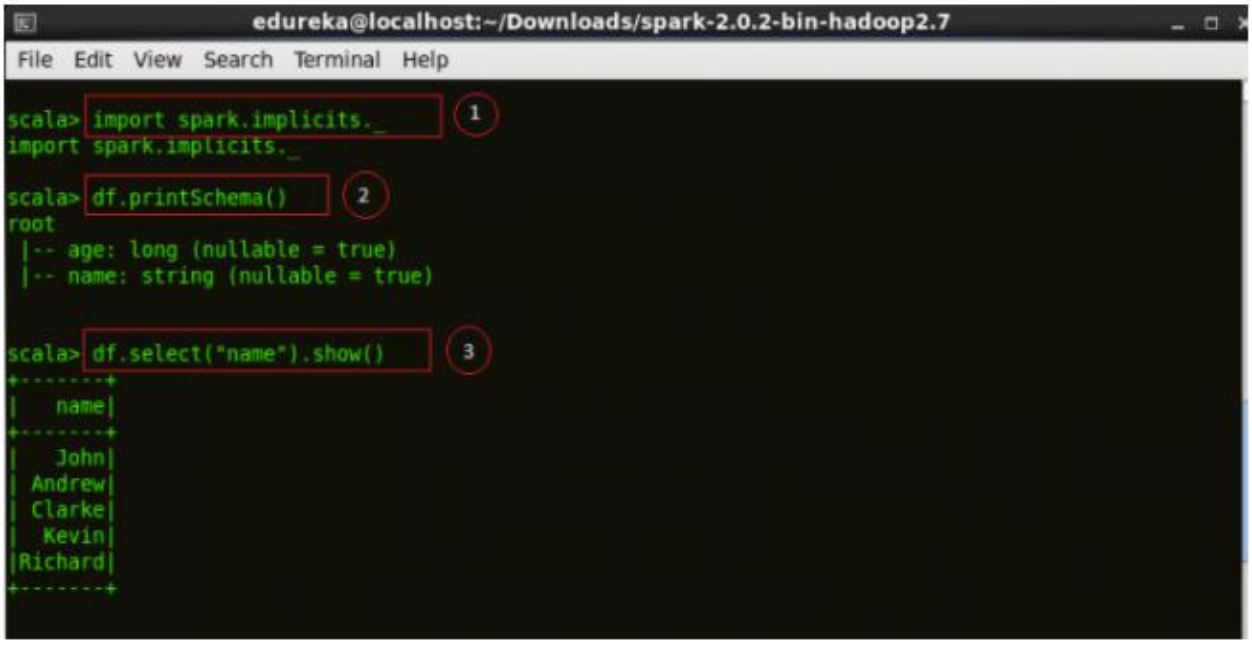


Figure 4.40 Schema of a DataFrame. [36]

Code explanation:

1. Displaying the DataFrame after incrementing everyone's age by two years.
2. We filter all the employees above age 30 and display the result.

```
1  df.select($"name", $"age" + 2).show()
2  df.filter($"age" > 30).show()
```

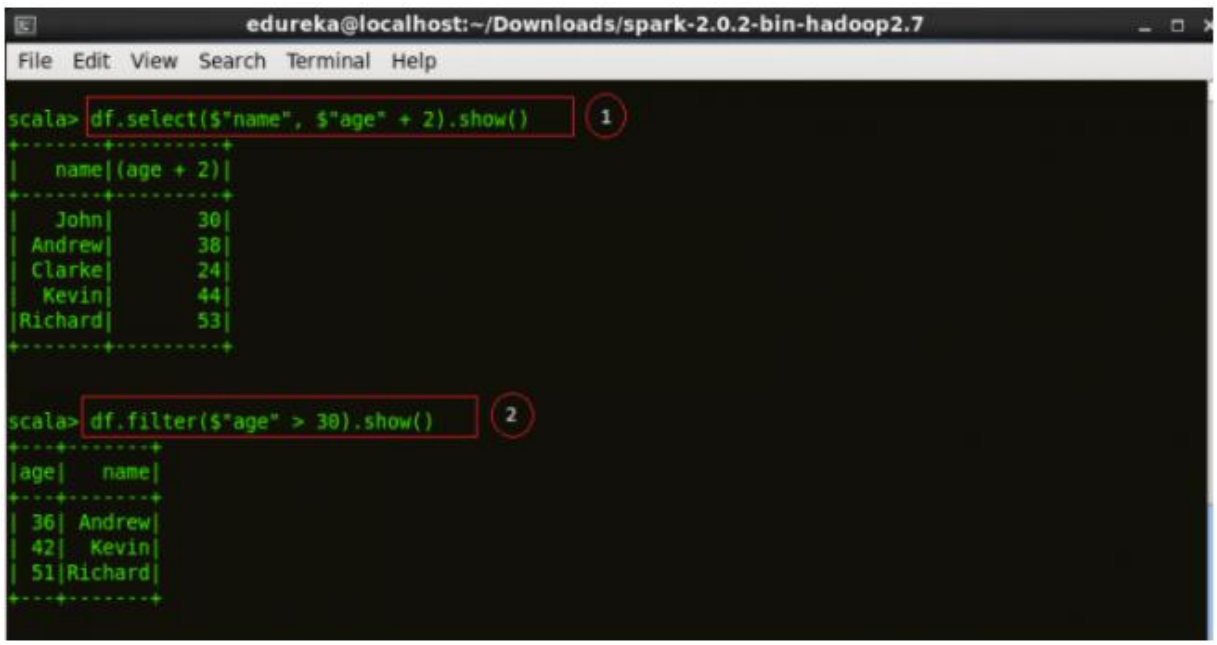


Figure 4.41 Basic SQL operations on employee.json. [36]

Code explanation:

1. Counting the number of people with the same ages. We use the 'groupBy' function for the same.
2. Creating a temporary view 'employee' of our 'df' DataFrame.
3. Perform a 'select' operation on our 'employee' view to display the table into 'sqlDF'.
4. Displaying the results of 'sqlDF'.

```
df.groupBy("age").count().show()
df.createOrReplaceTempView("employee")
val sqlDF = spark.sql("SELECT * FROM employee")
sqlDF.show()
```

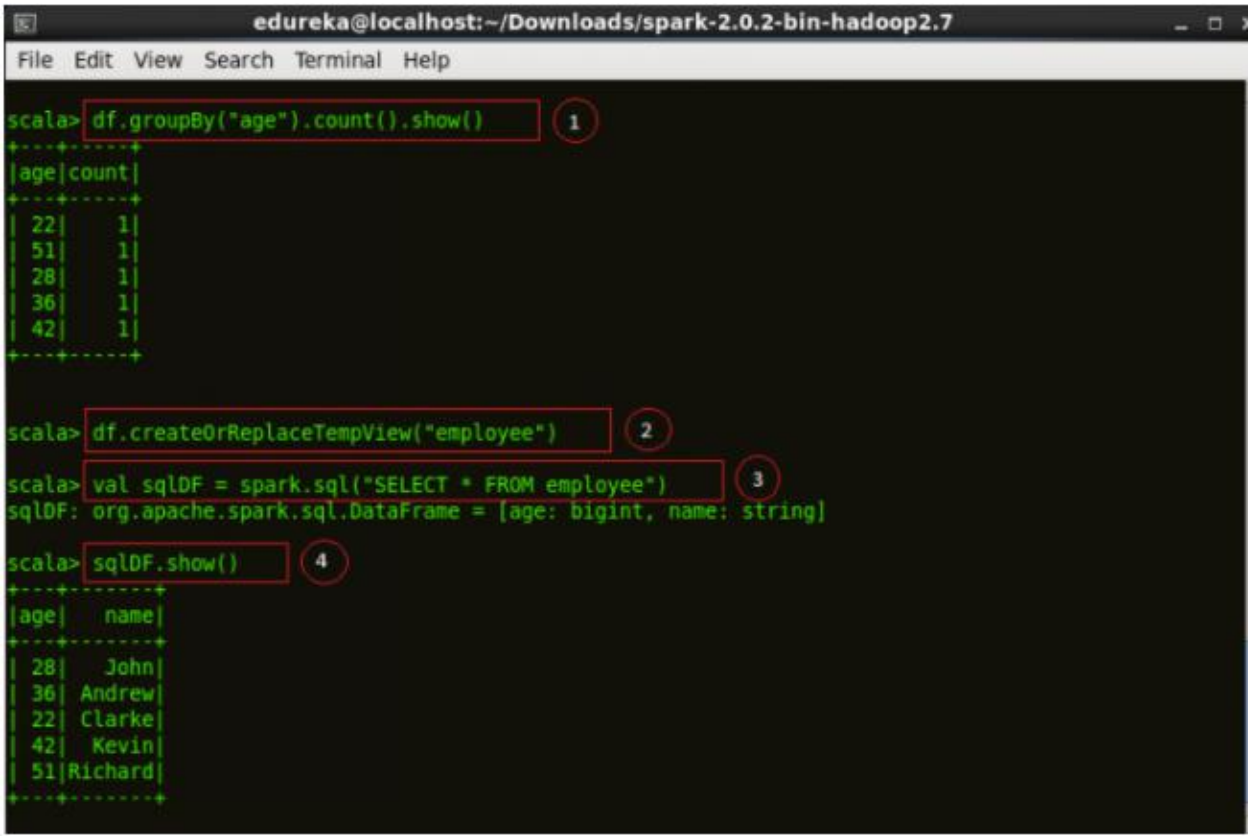


Figure 4.42 SQL operations on employee.json. [36]

4.4.7 Creating Datasets [36]

After understanding DataFrames, let us now move on to Dataset API. The below code creates a Dataset class in SparkSQL.

Code explanation:

1. Creating a class ‘Employee’ to store name and age of an employee.
2. Assigning a Dataset ‘caseClassDS’ to store the record of Andrew.
3. Displaying the Dataset ‘caseClassDS’.
4. Creating a primitive Dataset to demonstrate mapping of DataFrames into Datasets.
5. Assigning the above sequence into an array.

```
case class Employee(name: String, age: Long)
val caseClassDS = Seq(Employee("Andrew", 55)).toDS()
caseClassDS.show()
val primitiveDS = Seq(1, 2, 3).toDS
()primitiveDS.map(_ + 1).collect()
```

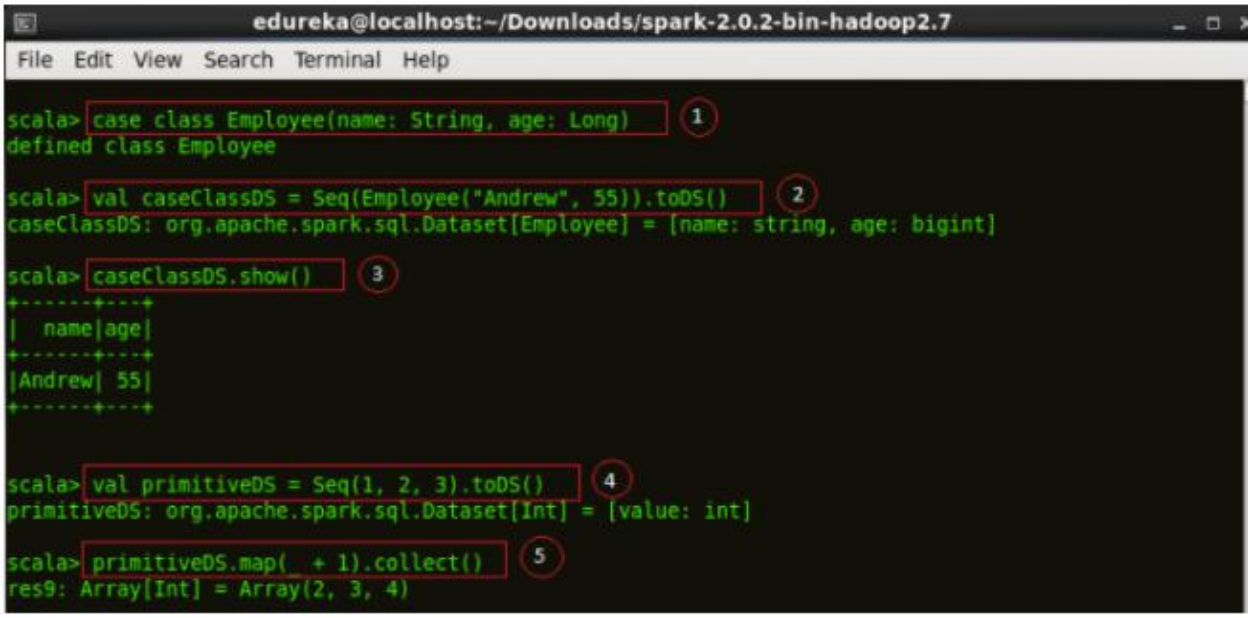


Figure 4.43 Creating a Dataset. [36]

Code explanation:

1. Setting the path to our JSON file ’employee.json’.
2. Creating a Dataset and from the file.
3. Displaying the contents of ’employeeDS’ Dataset.

```
val path = "examples/src/main/resources/employee.json"
val employeeDS = spark.read.json(path).as[Employee]
employeeDS.show()
```

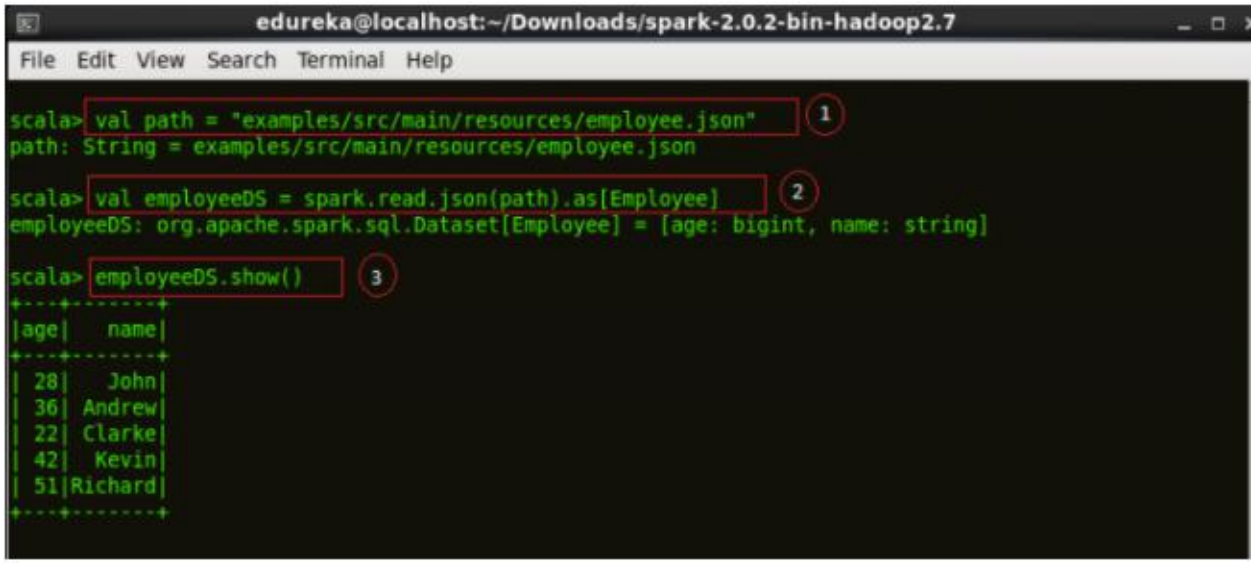


Figure 4.44 Creating a Dataset from a JSON file. [36]

4.4.8 Adding Schema To RDDs [36]

Spark introduces the concept of an RDD (Resilient Distributed Dataset), an immutable fault-tolerant, distributed collection of objects that can be operated on in parallel. An RDD can contain any type of object and is created by loading an external dataset or distributing a collection from the driver program.

Schema RDD is a RDD where you can run SQL on. It is more than SQL. It is a unified interface for structured data.

Code explanation:

1. Importing Expression Encoder for RDDs. RDDs are similar to Datasets but use encoders for serialization.
2. Importing Encoder library into the shell.
3. Importing the Implicts class into our ‘spark’ Session.
4. Creating an ’employeeDF’ DataFrame from ’employee.txt’ and mapping the columns based on the delimiter comma ‘,’ into a temporary view ’employee’.
5. Creating the temporary view ’employee’.
6. Defining a DataFrame ‘youngstersDF’ which will contain all the employees between the ages of 18 and 30.
7. Mapping the names from the RDD into ‘youngstersDF’ to display the names of youngsters.

```
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
```

```
import org.apache.spark.sql.Encoder
import spark.implicits._
val employeeDF =
spark.sparkContext.textFile("examples/src/main/resources/employee.txt").map(_.split(",")
).map(attributes => Employee(attributes(0), attributes(1).trim.toInt)).toDF()
employeeDF.createOrReplaceTempView("employee")
val youngstersDF = spark.sql("SELECT name, age FROM employee WHERE age
BETWEEN 18 AND 30")
youngstersDF.map(youngster => "Name: " + youngster(0)).show()
```

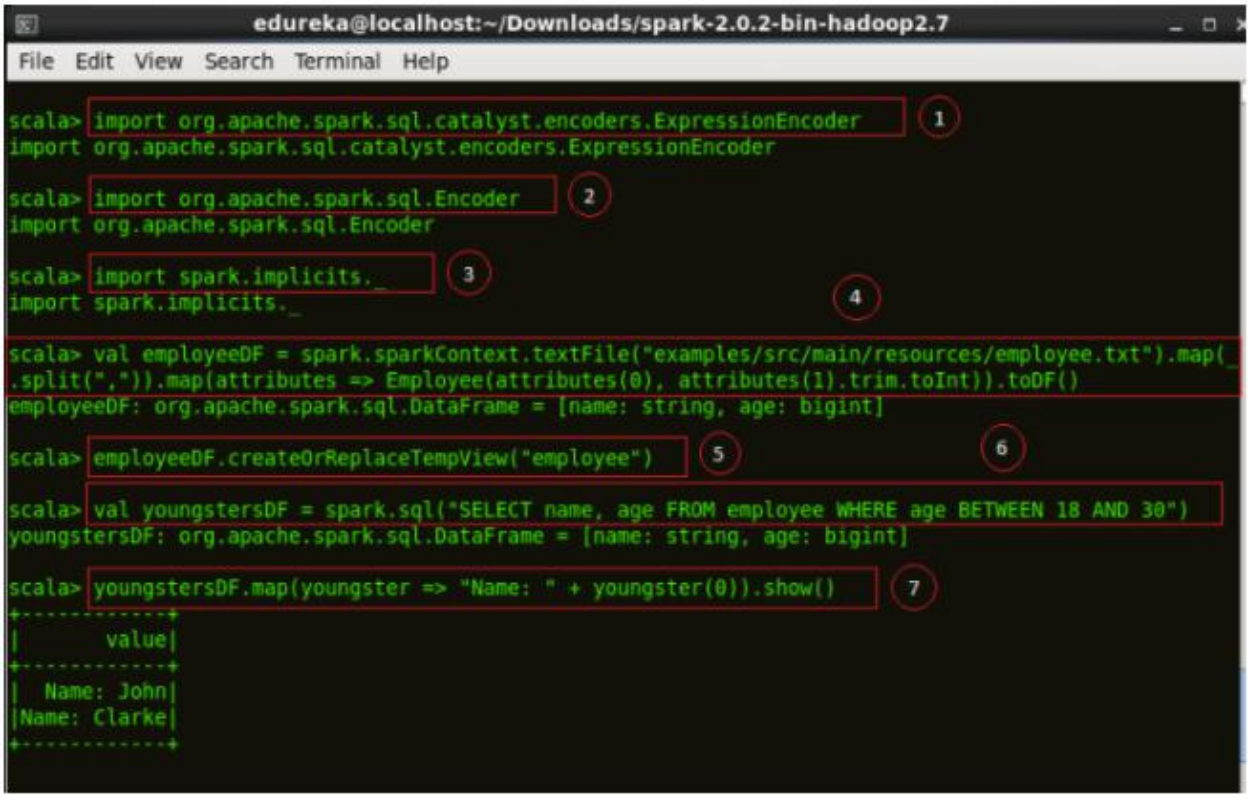


Figure 4.45 Creating a DataFrame for transformations. [36]

Code explanation:

1. Converting the mapped names into string for transformations.
2. Using the mapEncoder from Implicits class to map the names to the ages.
3. Mapping the names to the ages of our 'youngstersDF' DataFrame. The result is an array with names mapped to their respective ages.

```
youngstersDF.map(youngster => "Name: " + youngster.getAs[String]("name")).show()
implicit val mapEncoder = org.apache.spark.sql.Encoders.kryo[Map[String, Any]]
youngstersDF.map(youngster => youngster.getValuesMap[Any](List("name", "age"))).collect()
```

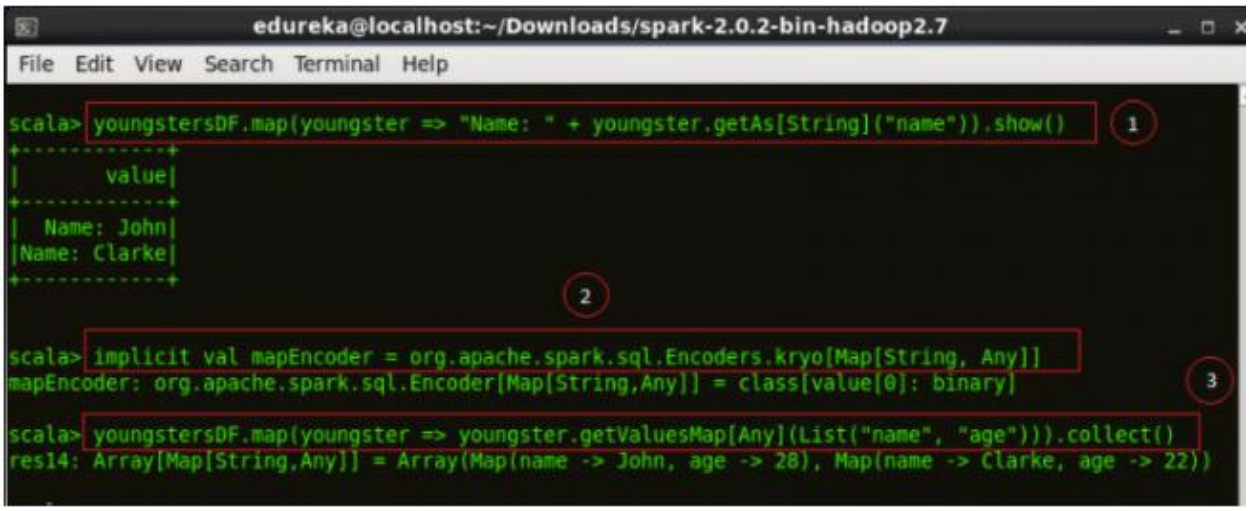


Figure 4.46 Mapping using DataFrames. [36]

RDDs support two types of operations:

- Transformations: These are the operations (such as map, filter, join, union, and so on) performed on an RDD which yield a new RDD containing the result.
- Actions: These are operations (such as reduce, count, first, and so on) that return a value after running a computation on an RDD.

Transformations in Spark are “lazy”, meaning that they do not compute their results right away. Instead, they just “remember” the operation to be performed and the dataset (e.g., file) to which the operation is to be performed. The transformations are computed only when an action is called and the result is returned to the driver program and stored as Directed Acyclic Graphs (DAG). This design enables Spark to run more efficiently. For example, if a big file was transformed in various ways and passed to first action, Spark would only process and return the result for the first line, rather than do the work for the entire file.

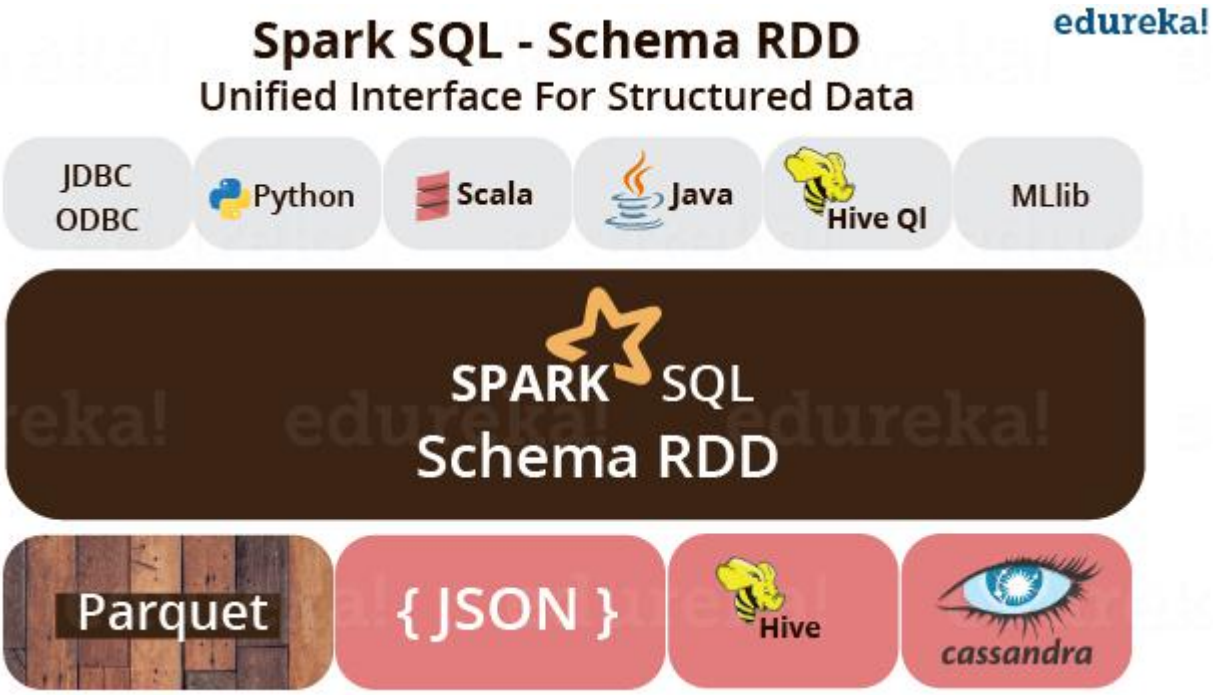


Figure 4.47 Ecosystem of Schema RDD in Spark SQL. [36]

By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory using the persist or cache method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it.

4.4.9 RDDs As Relations [36]

Resilient Distributed Datasets (RDDs) are distributed memory abstraction which lets programmers perform in-memory computations on large clusters in a fault tolerant manner. RDDs can be created from any data source. Eg: Scala collection, local file system, Hadoop, Amazon S3, HBase Table, etc.

Specifying Schema

Code explanation:

1. Importing the ‘types’ class into the Spark Shell.
2. Importing ‘Row’ class into the Spark Shell. Row is used in mapping RDD Schema.
3. Creating a RDD ’employeeRDD’ from the text file ’employee.txt’.
4. Defining the schema as “name age”. This is used to map the columns of the RDD.

5. Defining ‘fields’ RDD which will be the output after mapping the ’employeeRDD’ to the schema ‘schemaString’.
6. Obtaining the type of ‘fields’ RDD into ‘schema’.

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
val employeeRDD =
spark.sparkContext.textFile("examples/src/main/resources/employee.txt")
val schemaString = "name age"
val fields = schemaString.split(" ").map(fieldName => StructField(fieldName,
StringType, nullable = true))
val schema = StructType(fields)
```

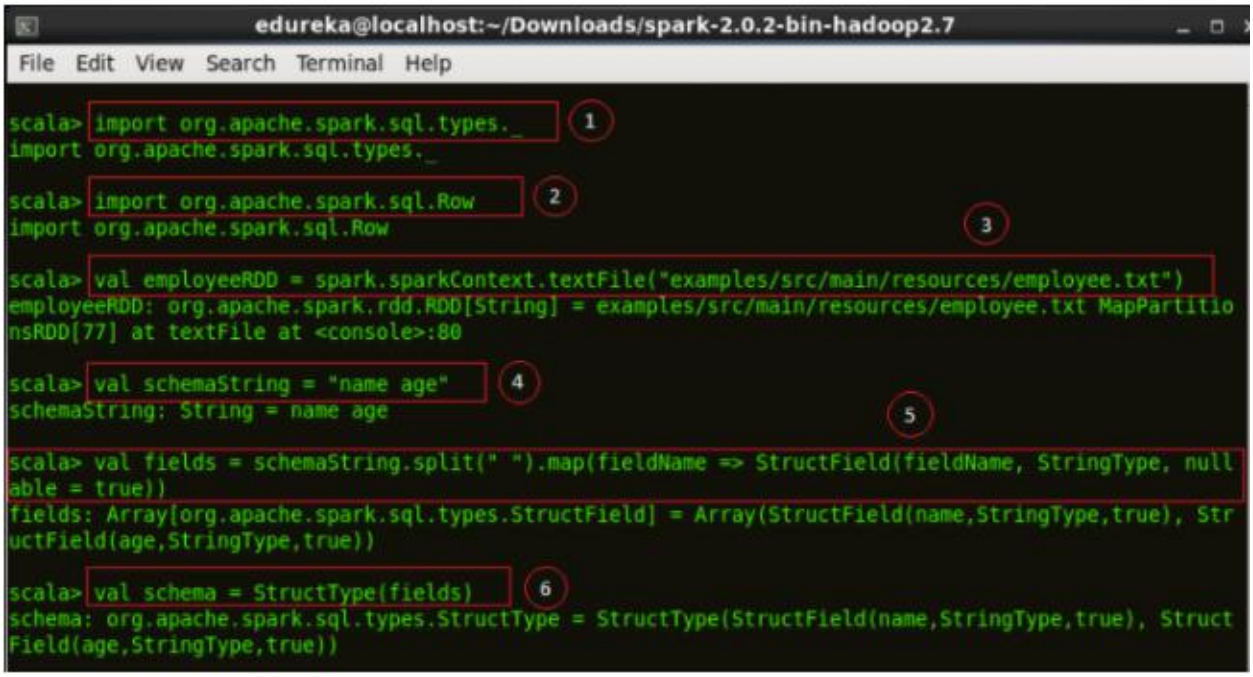


Figure 4.48 Specifying Schema for RDD transformation. [36]

Code explanation:

1. We now create a RDD called ‘rowRDD’ and transform the ’employeeRDD’ using the ‘map’ function into ‘rowRDD’.
2. We define a DataFrame ’employeeDF’ and store the RDD schema into it.
3. Creating a temporary view of ’employeeDF’ into ’employee’.
4. Performing the SQL operation on ’employee’ to display the contents of employee.
5. Displaying the names of the previous operation from the ’employee’ view.

```
1 val rowRDD = employeeRDD.map(_.split(",")).map(attributes => Row(attributes(0), attributes(1).trim))
2 val employeeDF = spark.createDataFrame(rowRDD, schema)
3 employeeDF.createOrReplaceTempView("employee")
4 val results = spark.sql("SELECT name FROM employee")
5 results.map(attributes => "Name: " + attributes(0)).show()
```

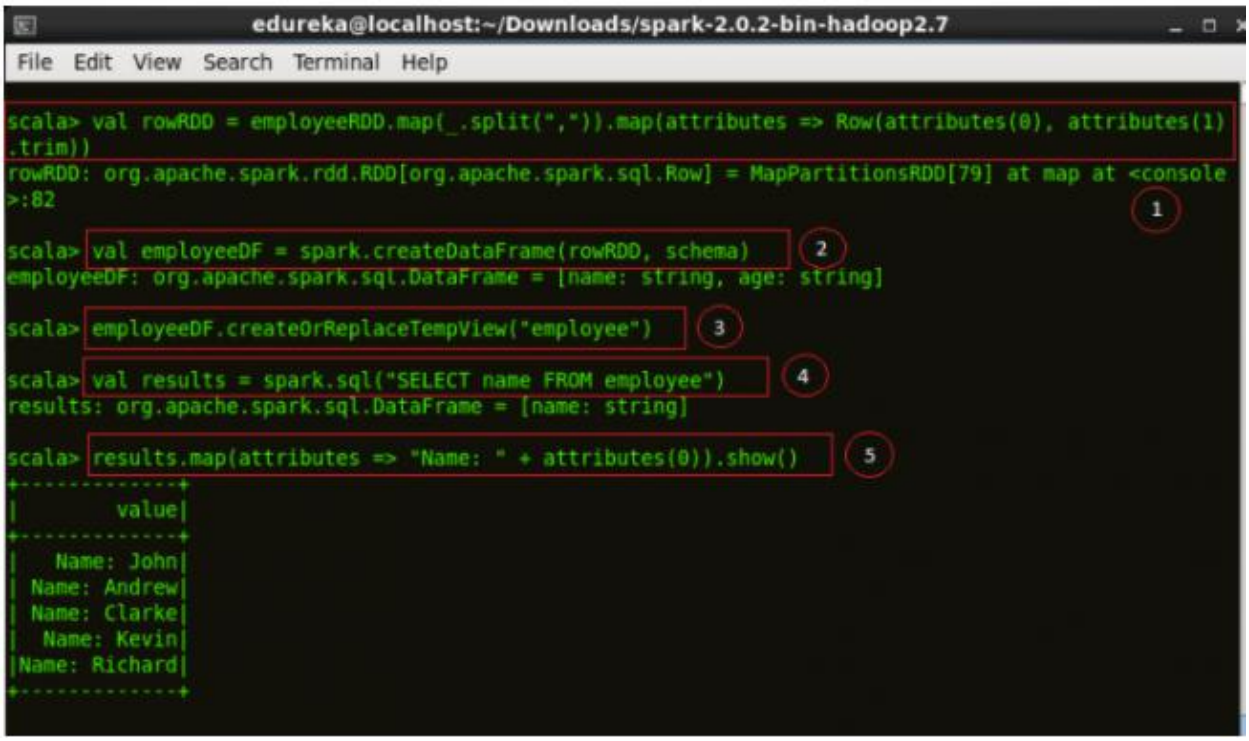


Figure 4.49 Result of RDD transformation. [36]

Even though RDDs are defined, they don't contain any data. The computation to create the data in a RDD is only done when the data is referenced. e.g. Caching results or writing out the RDD.

4.4.10 Caching Tables In-Memory [36]

Spark SQL caches tables using an in-memory columnar format:

1. Scan only required columns
2. Fewer allocated objects
3. Automatically selects best comparison

4.4.11 Loading Data Programmatically [36]

The below code will read employee.json file and create a DataFrame. We will then use it to create a Parquet file.

Code explanation:

1. Importing Implicits class into the shell.

2. Creating an ’employeeDF’ DataFrame from our ’employee.json’ file.

```
import spark.implicits._
val employeeDF = spark.read.json("examples/src/main/resources/employee.json")
```

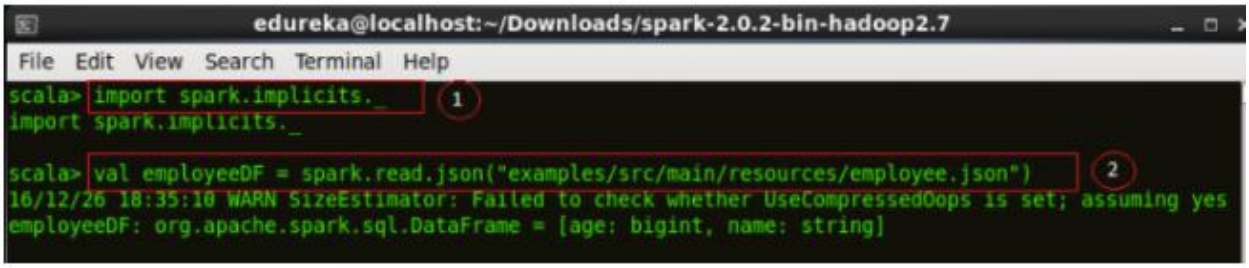


Figure 4.50 Loading a JSON file into DataFrame. [36]

Code explanation:

1. Creating a ‘parquetFile’ temporary view of our DataFrame.
2. Selecting the names of people between the ages of 18 and 30 from our Parquet file.
3. Displaying the result of the Spark SQL operation.

```
employeeDF.write.parquet("employee.parquet")
val parquetFileDF = spark.read.parquet("employee.parquet")
parquetFileDF.createOrReplaceTempView("parquetFile")
val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 18 AND 30")
namesDF.map(attributes => "Name: " + attributes(0)).show()
```

Figure 4.51 Displaying results from a Parquet DataFrame. [36]

4.4.12 JSON Datasets [36]

We will now work on JSON data. As Spark SQL supports JSON dataset, we create a DataFrame of employee.json. The schema of this DataFrame can be seen below. We then define a Youngster DataFrame and add all the employees between the ages of 18 and 30.

Code explanation:

1. Setting to path to our 'employee.json' file.
2. Creating a DataFrame 'employeeDF' from our JSON file.
3. Printing the schema of 'employeeDF'.
4. Creating a temporary view of the DataFrame into 'employee'.
5. Defining a DataFrame 'youngsterNamesDF' which stores the names of all the employees between the ages of 18 and 30 present in 'employee'.
6. Displaying the contents of our DataFrame.

```
1  val path = "examples/src/main/resources/employee.json"
2  val employeeDF = spark.read.json(path)
3  employeeDF.printSchema()
4  employeeDF.createOrReplaceTempView("employee")
5  val youngsterNamesDF = spark.sql("SELECT name FROM employee WHERE age BETWEEN 18 AND 30")
6  youngsterNamesDF.show()
```

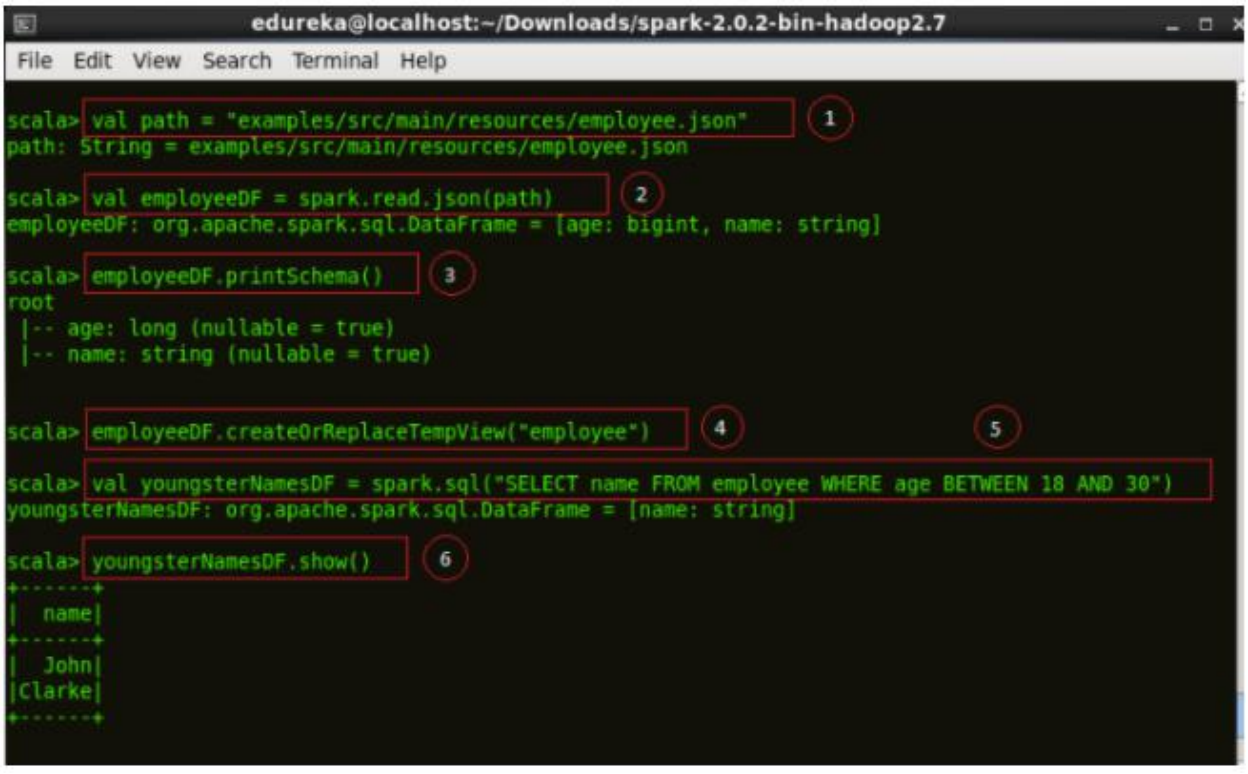


Figure 4.52 Operations on JSON Datasets. [36]

Code explanation:

1. Creating a RDD 'otherEmployeeRDD' which will store the content of employee George from New Delhi, Delhi.
2. Assigning the contents of 'otherEmployeeRDD' into 'otherEmployee'.
3. Displaying the contents of 'otherEmployee'.

```
val otherEmployeeRDD =
spark.sparkContext.makeRDD("""{"name":"George","address":{"city":"New
Delhi","state":"Delhi"}}""" :: Nil)
val otherEmployee = spark.read.json(otherEmployeeRDD)
otherEmployee.show()
```

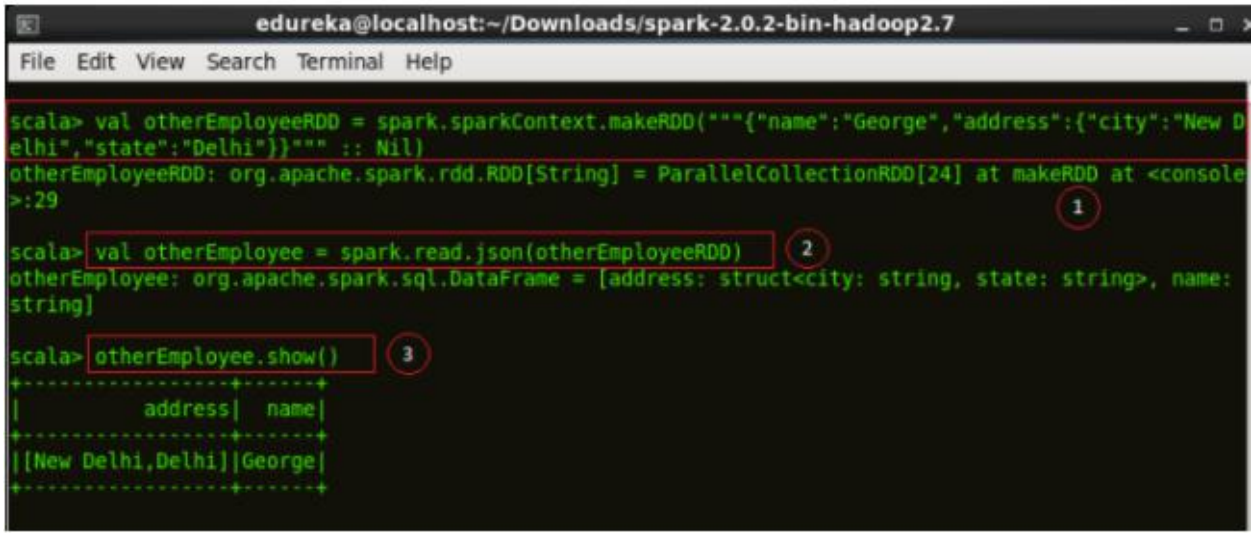


Figure 4.53 RDD transformations on JSON Dataset. [36]

4.4.13 Hive Tables [36]

We perform a Spark example using Hive tables.

Code explanation:

1. Importing 'Row' class into the Spark Shell. Row is used in mapping RDD Schema.
2. Importing Spark Session into the shell.
3. Creating a class 'Record' with attributes Int and String.
4. Setting the location of 'warehouseLocation' to Spark warehouse.
5. We now build a Spark Session 'spark' to demonstrate Hive example in Spark SQL.
6. Importing Implicits class into the shell.
7. Importing SQL library into the Spark Shell.

8. Creating a table ‘src’ with columns to store key and value.

```
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession
case class Record(key: Int, value: String)
val warehouseLocation = "spark-warehouse"
val spark = SparkSession.builder().appName("Spark Hive
Example").config("spark.sql.warehouse.dir",
warehouseLocation).enableHiveSupport().getOrCreate()
import spark.implicits._
import spark.sql
sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
```

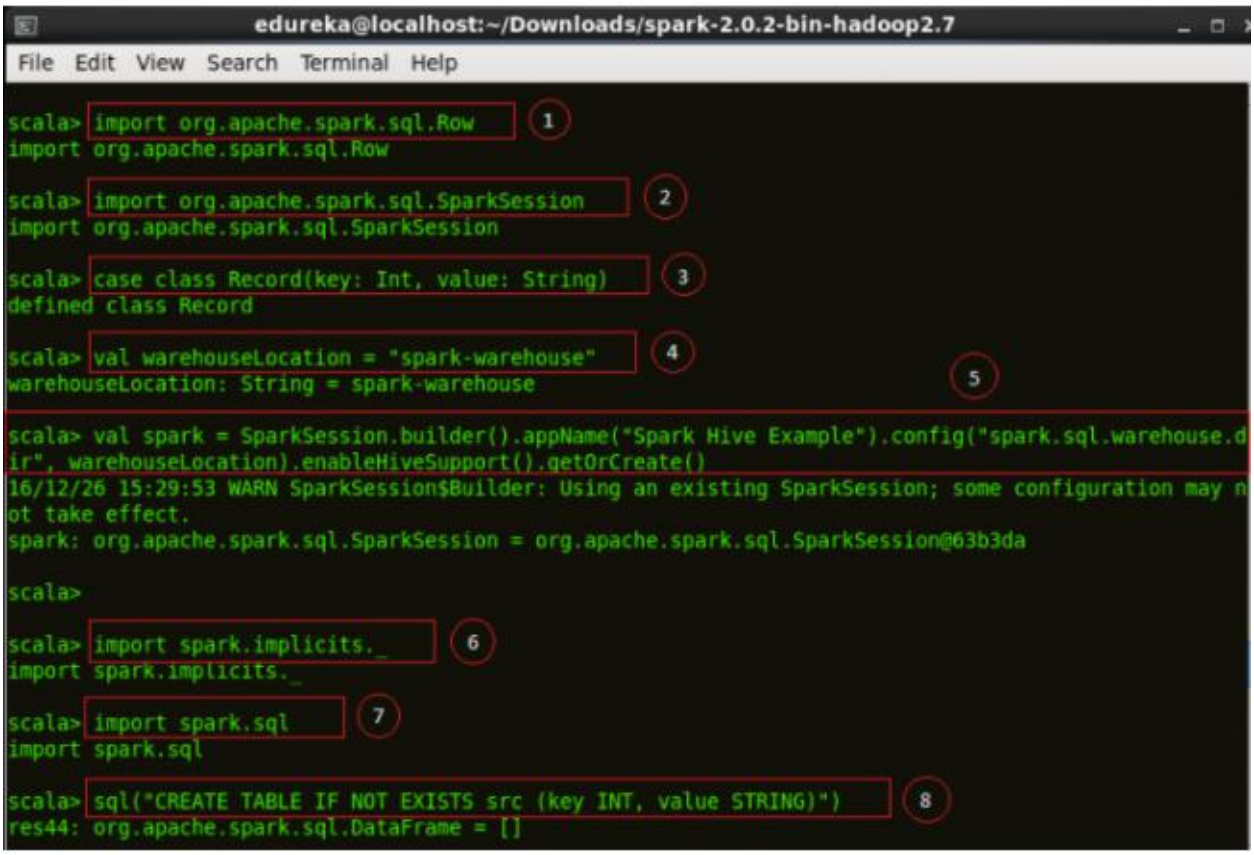


Figure 4.54 Building a Session for Hive. [36]

Code explanation:

1. We now load the data from the examples present in Spark directory into our table ‘src’.
2. The contents of ‘src’ is displayed below.

```
1 sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")
2 sql("SELECT * FROM src").show()
```

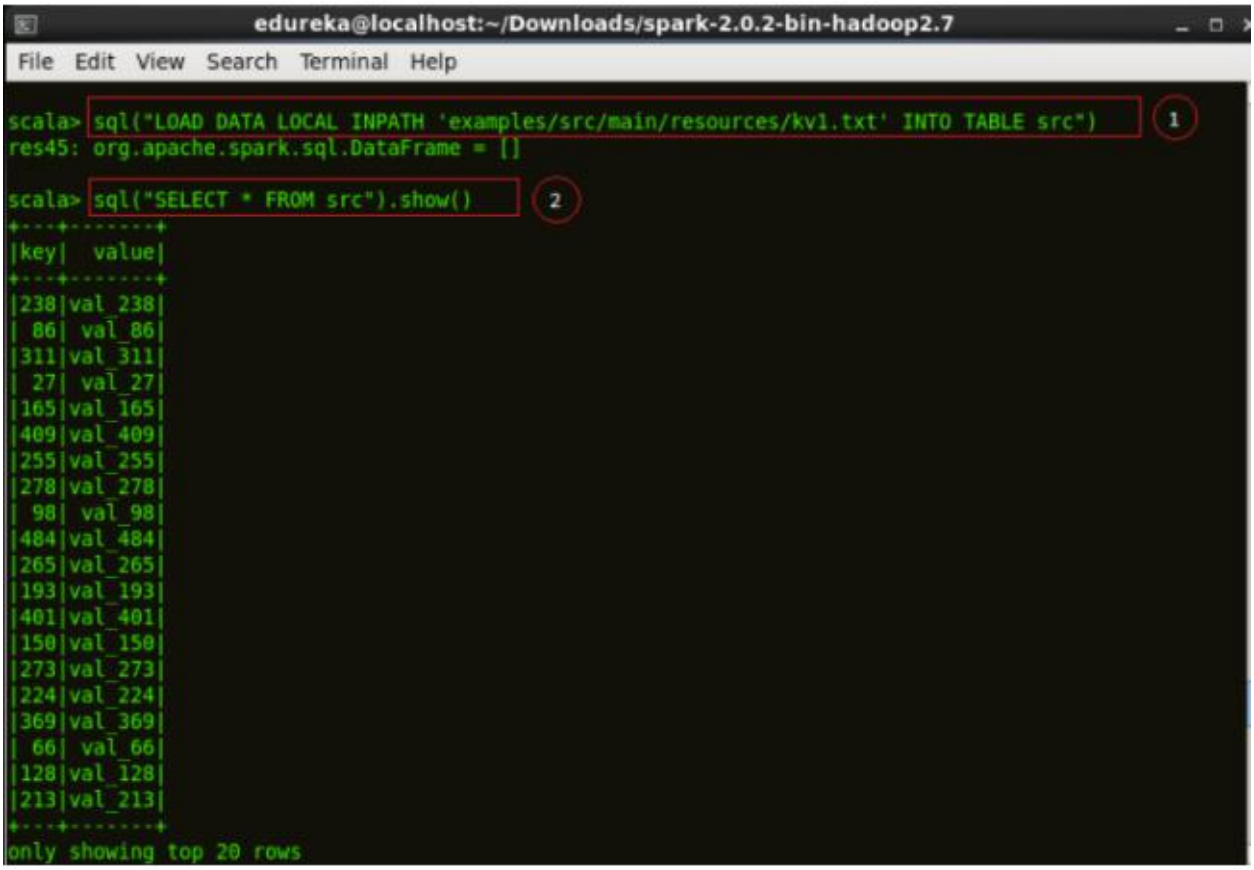


Figure 4.55 Selection using Hive tables. [36]

Code explanation:

1. We perform the 'count' operation to select the number of keys in 'src' table.
2. We now select all the records with 'key' value less than 10 and store it in the 'sqlDF' DataFrame.
3. Creating a Dataset 'stringDS' from 'sqlDF'.
4. Displaying the contents of 'stringDS' Dataset.

```
sql("SELECT COUNT(*) FROM src").show()
val sqlDF = sql("SELECT key, value FROM src WHERE key < 10 ORDER BY key")
val stringsDS = sqlDF.map {case Row(key: Int, value: String) => s"Key: $key, Value:
$value"}
stringsDS.show()
```

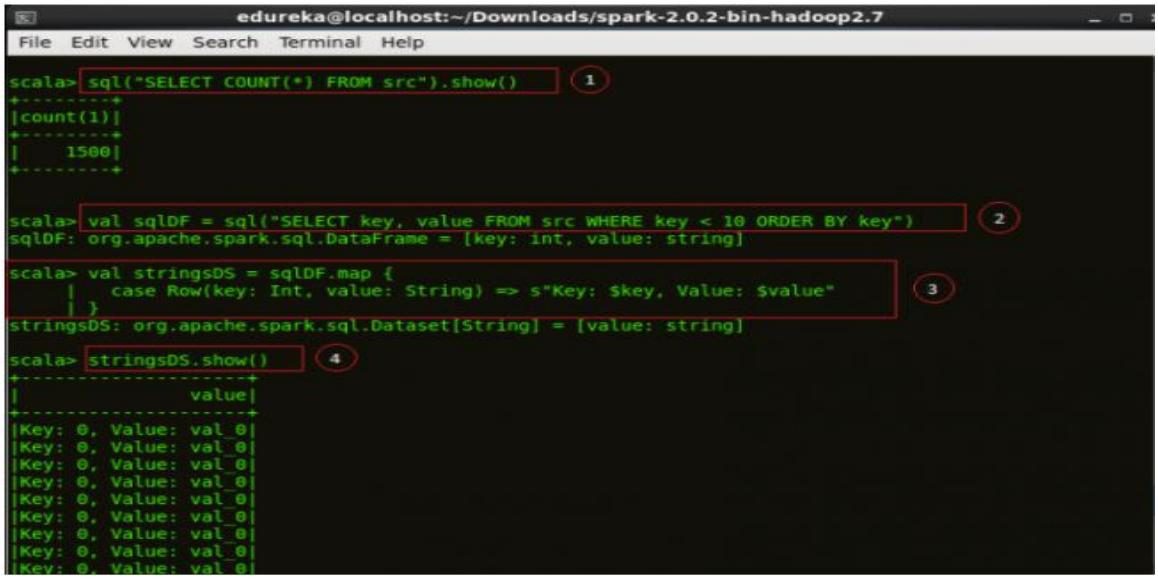


Figure 4.56 Creating DataFrames from Hive tables. [36]

Code explanation:

1. We create a DataFrame 'recordsDF' and store all the records with key values 1 to 100.
2. Create a temporary view 'records' of 'recordsDF' DataFrame.
3. Displaying the contents of the join of tables 'records' and 'src' with 'key' as the primary key.

```
1  val recordsDF = spark.createDataFrame((1 to 100).map(i => Record(i, s"val_$i")))
2  recordsDF.createOrReplaceTempView("records")
3  sql("SELECT * FROM records r JOIN src s ON r.key = s.key").show()
```

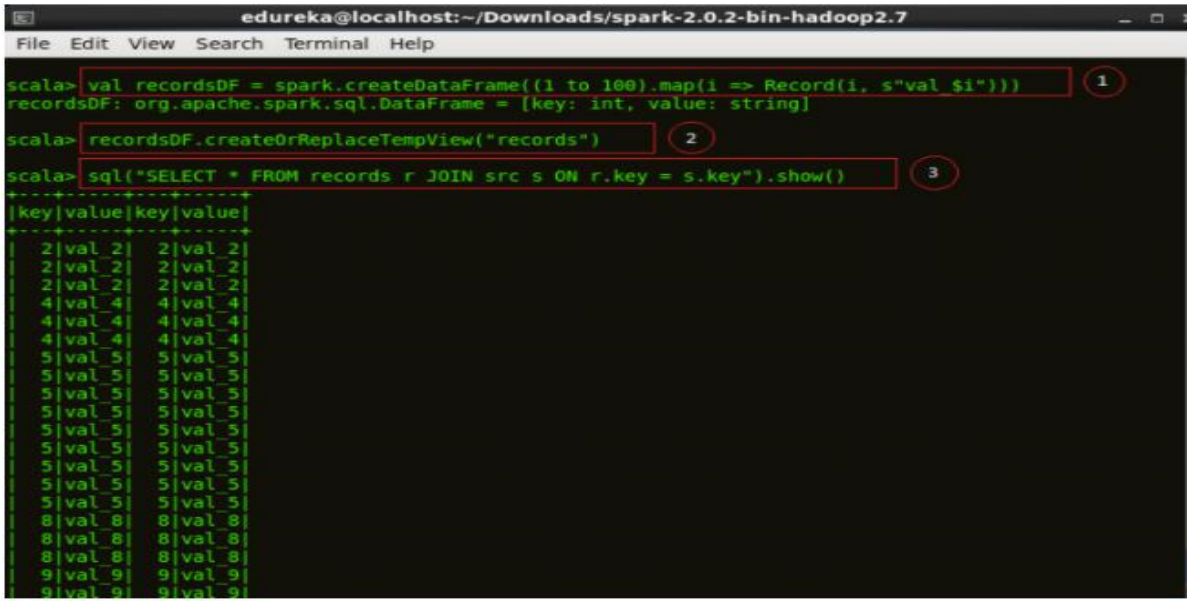


Figure 4.57 Recording the results of Hive operations. [36]

REFERENCES:

1. Agira corporate blog [Online]. Available:< http://www.agiratech.com/introduction-to-big-data-analytics/> [Accessed: 10-Feb-2018].
2. Tutorialspoints [Online]. Available:< https://www.tutorialspoint.com/big_data_analytics/index.htm> [Accessed: 10-Feb-2018]
3. Bart Baesens, Analytics in a Big Data World: The Essential Guide to Data Science and its Applications (Wiley and SAS Business Series), Wiley; 1 edition, May 19, 2014
4. J. Han and M. Kamber, Data Mining: Concepts and Techniques, 2nd ed. (Morgan Kaufmann, Waltham, MA, US, 2006); D. J. Hand, H. Mannila, and P. Smyth, Principles of Data Mining (MIT Press, Cambridge, Massachusetts, London, England, 2001); P. N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining (Pearson, Upper Saddle River, New Jersey, US, 2006).
5. D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, "Performance of Classification Models from a User Perspective." Special issue, Decision Support Systems 51, no. 4 (2011): 782–793.
6. J. Banasik, J. N. Crook, and L. C. Thomas, "Sample Selection Bias in Credit Scoring Models" in Proceedings of the Seventh Conference on Credit Scoring and Credit Control (Edinburgh University, 2001).
7. R. J. A. Little and D. B. Rubin, Statistical Analysis with Missing Data (Wiley-Interscience, Hoboken, New Jersey, 2002).
8. Bernard Marr, "Big Data Using Smart Big Data, Analytics and Metrics to Make Better Decisions and Improve Performance", Wiley, 2015
9. Mayer-Schonberger, V. and Cukier, K., "Big Data: A Revolution That Will Transform How We Live, Work and Think", London: John Murray Publishers, 2013
10. Neilson, J.P. and Mistry, R.T., "Fetal electrocardiogram plus heart rate recording for fetal monitoring during labour", Cochrane Database of Systematic Reviews (2), 2000
11. Dekker, J.M., Schouten, E.G., Klootwijk, P., Pool, J., Swenne, C.A. and Kromhout, D., "Heart rate variability from short electrocardiographic recordings predicts mortality from all causes in Middle-aged and elderly men", The Zutphen Study, American Journal of Epidemiology 145 (10), 1997
12. IACP Centre for Social Media Fun Facts http://www.iacpsocialmedia.org/ Resources/FunFacts.aspx
13. BBC Two (2014) Bang goes the Theory, May 2014, Series 8: Big Data.
14. SAS Whitepaper, "Big Data meets Big Data Analytics: Three key technologies for extracting real-time business value from the Big Data that threatens to overwhelm traditional computing architectures", 2012
15. IDC (2014) The Digital Universe Study, April 2014. Sponsored by EMC2
16. Chui M, Loffler M, and Roberts R, "The Internet of Things". ¨ McKinsey Quarterly, March 2010.
17. Kosinski, M., Stillwell, D. and Graepel, T. (2013) Private traits and attributes are predictable from digital records of human behavior. Published online: http://www.pnas.org/content/early
18. David Loshin, "Big Data Analytics: From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph", Morgan Kaufmann, 2013

19. Hanson J. An introduction to the Hadoop distributed file system, accessed via http://www.ibm.com/developerworks/library/wa-introhdfs.
20. Apache's HDFS Architecture Guide, accessed via http://hadoop.apache.org/docs/stable/hdfs_design.html.
21. Murthy A. Introducing Apache Hadoop YARN, accessed via http://hortonworks.com/blog/introducing-apache-hadoop-yarn/.
22. Zookeeper, accessed via http://zookeeper.apache.org/.
23. Apache, accessed via http://hive.apache.org/.
24. Pig, accessed via http://pig.apache.org/.
25. Mahout, accesed via http://mahout.apache.org/.
26. Thomas Erl, Wajid Khattak, and Paul Buhler, "Big Data Fundamentals, Concepts, Drivers & Techniques", Prentice Hall, 2016
27. Sean Owen,‵ Robin Anil,‵ Ted Dunning,‵ Ellen Friedman, "Mahout in Action", Manning Publications, 2011
28. Practical eCommerce, "10 Questions on Product Recommendations," http://mng.bz/b6A5
29. Google Blogoscoped, "Overall Number of Picasa Photos" (March 12, 2007), http://blogoscoped.com/archive/2007-03-12-n67.html
30. TutorialsPoint [Online], Available: https://www.tutorialspoint.com/mahout/mahout_introduction.htm, [Accessed: 24-Feb-2018]
31. Wisdomjobs [Online], Available: https://www.wisdomjobs.com/e-university/hadoop-tutorial-484/a-brief-history-of-hadoop-14745.html [Accessed: 26-Feb-2018]
32. Edureka [Online], Available: https://www.edureka.co/blog/hadoop-tutorial/ [Accessed: 26-Feb-2018]
33. bmc [Online], Available: http://www.bmc.com/guides/hadoop-examples.html [Accessed: 26-Feb-2018]
34. bmc [Online], Available: http://www.bmc.com/guides/hadoop-benefits-business-case.html [Accessed: 26-Feb-2018]
35. JavaTPoint [Online], Available: https://www.javatpoint.com/hadoop-tutorial [Accessed: 27-Feb-2018]
36. Edureka [Online], Available: https://www.edureka.co/blog/spark-tutorial/ [Accessed: 04-March-2018]
37. http://www.scala-lang.org/ [Accessed: 04-March-2018]
38. http://spark.apache.org/downloads.html [Accessed: 04-March-2018]
39. https://drive.google.com/file/d/0B7Yoht-ttAeuWGd5SC1LcVZUbkk/view [Accessed: 04-March-2018]
40. https://docs.google.com/forms/d/e/1FAIpQLScJyoUgebjUlRyG9JrXY-aF9M4P2ZuMhFW7pzXzWaZ2IMEcxw/viewform [Accessed: 04-March-2018]
41. https://docs.google.com/forms/d/e/1FAIpQLSfMzE3sUoIASRbVanVwdlUX2h-1vXiksSxusHjSqhj_CR6RhQ/viewform [Accessed: 04-March-2018]
42. https://docs.google.com/forms/d/e/1FAIpQLSdPiCNiObU145181i0IyBEmHQtk5V09Wyq7F0ZYVyx1H-faYA/viewform [Accessed: 04-March-2018]