# Big Data Processing Using Spark and Machine Learning Library (MLlib) on Multinode Hadoop Cluster

MOHAMMAD TAQI S/O HASSAN ALI

SESSION 2014/2015

FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

JANUARY 2015

# Big Data Processing Using Spark and Machine Learning Library (MLlib) on Multinode Hadoop Cluster

BY

MOHAMMAD TAQI S/O HASSAN ALI

SESSION 2014/2015

THIS PROJECT REPORT IS PREPARED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
IN PARTIAL FULFILLMENT
FOR
BACHELOR OF INFORMATION TECHNOLOGY (HONS)
SOFTWARE ENGINEERING
FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

JANUARY 2015

# Declaration

I hereby declare that the work have been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

*Mohammad Taqi S/O Hassan Ali*
Faculty of Computing & Informatics
Multimedia University
Date: $15^{th}$ January 2015

# Acknowledgements

It is a pleasure to thank those who made this thesis possible. I owe my deepest gratitude to my project supervisor, Mr. Wan Ruslan Yusoff without his enthusiasm, his inspiration, and his ability to explain things in a clear and precise manner, this project would not have been possible. I feel extremely fortunate to have him as my supervisor and life advisor. I am further in debt to him for guiding me in various kinds of academic and professional activities. He has supported me in infinite number of ways from giving keys to Grid lab to taking us for dinner and sharing his experience in his valuable professional careers. He is not just a project supervisor, but a friend, whose support extends beyond the project.

Then I am grateful of my family and their financial support for the last four years of my stay in Malaysia, because of their support I could stay in one of the best places and I never faced any financial bar from university. My father prayers was always with me during all exams, assignments, and obstacles and I am grateful of him.

At the end I would like to thank all my friends whom their intellectual support and their ideas helped me a lot during this project.

# Contents

# List of Figures

# List of Tables

| **Acronym** | **What It Stands For** |
|---|---|
| MMU | Multimedia University |
| FCI | Faculty of Computing & Informatics |
| FYP | Final Year Project |
| RDBMS | Relational Database Management System |
| API | Application Program Interface |
| GFS | Google File System |
| MLlib | Machine Learning Library |
| HDFS | Hadoop Distributed File System |
| HTML | Hyper Text Markup Language |
| XML | Extensible Markup Language |
| SQL | Structure Query Language |
| CCTV | Closed-circuit television |
| NCDC | National Climate Data Center |
| GPS | Global Positioning System |
| GPRS | General packet radio service |
| I/O | Input & Output |
| 3V's | Volume, Velocity, Variety |
| YARN | Yet Another Resource Negotiator |
| AMPLab | Algorithms, Machines, and People Lab |
| Amazon S3 | Amazon Simple Storage System |
| kNN | K-Nearest Neighbors |
| NB | Naive Bayes |
| SVM | Support Vector Machine |
| SSH | Secure Shell |
| JDK | Java Development Kit |
| GNU | GNU's Not Unix |

Table 1: Abbreviations and Acronyms

| **Quantity** | **Unit** | **Equals** |
|---|---|---|
| 1 | Bit | Value of 0 or 1 |
| 1 | Byte | 8 Bits |
| 1 | KB (Kilobyte) | 1024 Bytes |
| 1 | MB (Megabyte) | 1024 KB |
| 1 | GB (Gigabyte) | 1024 MB |
| 1 | TB (Terabyte) | 1024 GB |
| 1 | PB (Petabyte) | 1024 TB |
| 1 | EB (Exabyte) | 1024 PB |
| 1 | ZB (Zettabyte) | 1024 EB |
| 1 | YB (Yottabyte) | 1024 ZB |

TABLE 2: Major Data Storage Units

# Management Summary

This project is titled ”Big Data Processing Using Spark and Machine Learning Library on Hadoop Cluster”. In this project we studied the methods in Big Data processing on a machine learning problem. Hadoop is an open-source framework for large-scale data storage and distributed computing, built on the MapReduce model. Spark is an application built on top of Hadoop for processing data in the computer's memory instead of hard disk storage. The Spark suite of applications also includes a machine learning library (MLlib).

This project started with understanding of Big Data, installation of Apache Hadoop and Spark on a single machine as well as on a multi-node cluster. We used Hadoop-2.4.1 and Spark-1.1.1 for both single machine and multi-node cluster.

We studied the challenges of processing Big Data, the tools to tackle these challenges, and limitations of these tools. Although the Hadoop and MapReduce combination is a good platform for data storage and batch processing, not all programs can be formed to use the MapReduce processing model. MapReduce is a very specific processing model, whereas Spark can be used for general processing.

In Phase 1 of this project, we learned about Hadoop Big Data processing technology and overcoming the complexity of its configuration. We finally managed to prove our successful installation and configuration of Hadoop and Spark by a running simple word count application, written in form of MapReduce, on both the single machine and multi-node cluster.

In Phase 2, we chose an appropriate benchmark census data from the University of California Irvine machine learning data repository for our machine learning classification problem. We studied various machine learning algorithms for both supervised and unsupervised machine learning. In classification, our goal is to be able to classify individuals' salaries greater than 50K against those less than or

equal to 50K. We were able to calculate the error rate, misclassified items, percentage of correct outcome of our classifier on the benchmark data. The algorithm that we chose to do classification (supervised learning) is logistic regression.

Our findings in this project are satisfactory. For individuals with salaries less than or equal to 50K, our classifier prediction is 95 percent correct, while for those with salaries greater than 50K, it is just 42 percent correct. We believed that the low prediction accuracy for the group with salaries greater than 50K are attributed to the simple assumptions made in the logistic regression stochastic (probability) model and the wide variations (dispersion in values) for its features.

# Chapter 1

# Introduction

We live in a world that technology is a need of survival. We can no longer ignore the power of computers and technologies in our everyday lives. In the early days, computers were only used as a tool of processing and as an assistant to humans. Nowadays a single computer machine can handle tasks that thousands of men can't even think of doing it.

Using this powerful machines we had started to generate and store massive data and information which couldn't be captured and recorded in traditional ways. We could no longer print all data and information in papers, books, files, and folders. Hence, databases came to life and with the help of large capacity hard disks we could capture the relational data and get use of them.

RDBMS (Relational Database Management System) was so far one of the best tools to store and retrieve these data. The data were stored in form of related tables but we could only store structured data. These data were mostly generated by computer professionals and they were only useful for their respective organizations.

Then drastic changes came with the emergence of social networks and smart devices(like phones and tablet and smart-watches, glasses, etc) into the world of IT

(Information Technology). This resulted that normal computer users participating in data generation.

Traditional databases were no longer good solutions to store these data, since these newly generated data were not structured and could not be placed in tables. Organizations and giant enterprises needed to capture these flood of information and get use of them to serve their customers in better ways.

Advertisers and commercial companies were among those who needed these information in order to study consumer behaviors and target their products in their best offerings.

As a result, giant computer organization, Google Inc., which nowadays appear to be the pioneer in everything, in 2003 and 2004 released two academic papers describing Google technology:

- Google File System (GFS) (`http://research.google.com/archive/gfs.html` )
- MapReduce Algorithm (`http://research.google.com/archive/mapreduce.html` )

The two together provided a platform for processing data on a very large scale and in a highly efficient manner.[1]

At the same time, Doug Cutting was working on Nutch[1] open source search engine took advantage of these paper and created Hadoop platform with the support of Yahoo! Doug started working on the implementations of these Google systems,

[1]Apache Nutch is a highly extensible and scalable open source web crawler software project.[2]

and Hadoop was soon born, firstly as a subproject of Lucene[2] and soon was its own top-level project within the Apache open source foundation.[1]

At its core, therefore, Hadoop is an open source platform that provides implementations of both the MapReduce and GFS technologies and allows the processing of very large data sets across clusters of low-cost commodity hardware.[1]

## 1.1 Project

Hadoop is an open-source framework for large-scale data storage and distributed computing, built on the MapReduce model.

Apache Hadoop, a nine-year-old open-source data-processing platform first used by Internet giants including Yahoo and Facebook, leads the big-data revolution. Hadoop has provided a feasible approach to streamline the tedious task of data storage and query by using the cheap commodity machines. The general concept of Hadoop is bringing the process to data in appose to the traditional concept of bringing data to process.

In this project, we will use Apache Spark (`http://spark.apache.org/`), which is a fast and general engine for large-scale data processing, and more specifically its Machine Learning Library (`http://spark.apache.org/mllib/`) running on a multi-node Hadoop cluster to perform online big data processing.

[2]Apache Lucene is a free/open source information retrieval software library, originally written in Java by Doug Cutting.[3]

## 1.2 Objectives

By the end of this project we will cover the following objectives

- Understanding Big Data in general.
- Understanding the current challenges of Big Data.
- Learn why Hadoop can be a good solution to these challenges.
- Identify the drawbacks of Hadoop and identifying the types of application that can not be formed in MapReduce model.
- Solution for Hadoop's drawbacks (Apache Spark and MLlib).
- Learn the concepts of machine learning and major classification algorithms.
- Solving a classification problem as an example of applications that can not be presented in form of MapReduce and solving it using Spark and MLlib API.
- Make conclusion based on findings of running applications on both MapReduce form and Spark form.

## 1.3 Motivations

- Data processing in the past decade was a tedious and boring task since there was not much innovation in data processing and all the data were processed in relational databases and there was not much opportunity to improve.
- Since we are living in the world of data so in order to live better in this world we would better know how to work and get use of these data.
- Data scientist are on high demand and getting higher salaries than other IT professionals.

- Data is becoming more ubiquitous as well as more understandable. And big decisions in high-level management require a deep and clear understanding of data processing and analytics.

## 1.4 Scope of This Project

1. Studying books, journals, articles, and other literature reviews to get a good understanding of Big Data and current state of it.
2. Identify the challenges of Big Data and what are different tools that can handle these challenges and understand why Hadoop is best choice among them.
3. Learn the infrastructure of Hadoop and its component to grasp a fair understanding of how Hadoop solves the challenges of Big Data.
4. Applying our understanding of Hadoop by installing and configuring different versions of Hadoop on different environments; such as, single machine (laptop) and Clusterrocks, 10 nodes Rock Cluster.
5. Observing the drawback of Hadoop by running simple word count, which is written in form of MapReduce application, on clusterrocks and recording the result.
6. Studying books, journals, articles, and other literature reviews to get a good understanding of machine learning and different classification algorithms.
7. Learning Spark infrastructure and its MLlib API to be able to apply a classification algorithm on census dataset.
8. Find an appropriate classification problem to solve using Spark and MLlib.
9. Develop a program to solve the problem found in number 8 and run it on multinode Clusterrocks environment.

10. Finally draw a conclusion based on our findings of running the program developed on item 9.

# Chapter 2

# Literature Review

# Part One

## 2.1 Introduction to Big Data

### 2.1.1 What is Big Data?

Basically Big Data doesn't have a specific definition but we can define some characteristics for it.

Big Data can refer to a collection of data that is so large and complex to process with traditional data management tools. Big Data is used to refer to a set of structured (e.g. relational data), semi-structured (e.g. XML, and HTML), and unstructured (e.g: Email, Picture, Video, Log files, signals, etc) data.

By the end of 2016, Cisco estimates that the annual global data traffic will reach 6.6 zettabytes.[4]

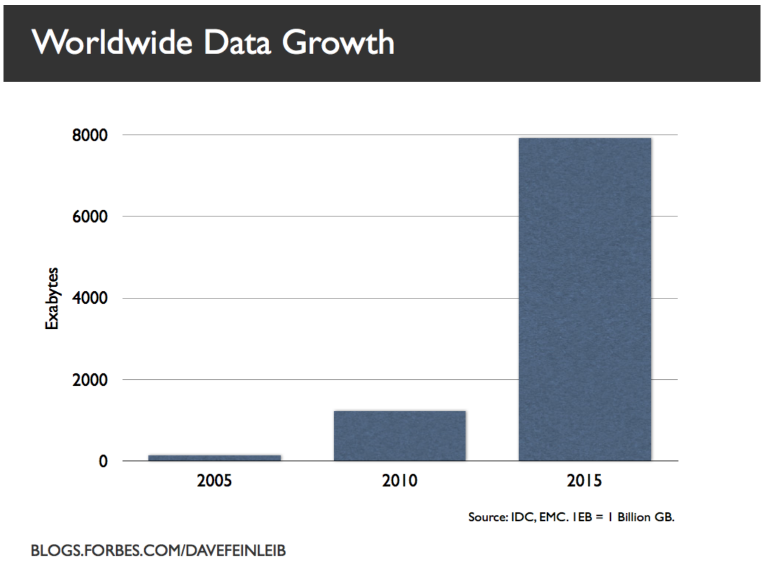


FIGURE 2.1: Data Growth Diagram

### 2.1.2 Who generates Big Data

There are many factors involving in Big Data generation such as: sensors, CCTVs, Social Networks (e.g. Facebook, Twitter, etc), online shopping, airlines, NCDC (National Climate Data Center), and many more.

With the emergence of smart phones these data generation even become faster because now we can start doing almost everything with our smart phones from making a simple calls to hacking to difficult systems.

Whether you are posting a status on your Facebook wall, listening to music on Spotify, twitting on Twitter, checking some product on Amazon, using Google Map to find nearest KFC to you, checking your bank statement, or any other activity you can think of it with your digital device on Internet; you are adding up on the amount of Big Data.

Basically We can categorize those who generate data in three groups:

I. Hardwares that automatically generate data such as Sensors, CCTV, Trackers, GPS, GPRS, etc.modern high-energy physics experiments, such as DZero[1], typically generate more than one terabyte of data per day.[5]

II. Organizations that generate data in an enormous speed. Giant IT companies are participating more than others in these type of data generations for example The famous social network website, Facebook, serves 570 billion page-views, stores 3 billion new photos, and manages 25 billion pieces of content monthly. Google Earth uses 70.5 TB: 70 TB for the raw imagery and 500 GB for the index data in 2006. [5]

III. People who are using modern technologies. Users of computers and smart phones are generating data by visiting websites, posting their photos and videos in social networks, tweeting about their daily activities, etc. for

[1]http://www-d0.fnal.gov/

example tweeter one of the most popular websites has 271 million monthly active users, gets 500 million tweets a day.[2]

### 2.1.3 Challenges of Big Data

Big Data is hard to process and handle due to three characteristics of it which is known as three V's of Big Data and they are as follow:

1. **Volume:** The first characteristic of Big Data, which is "Volume", refers to the quantity of data that is being manipulated and analyzed in order to obtain the desired results. It is challenging to process and analyze big volume of data to get the best result using less resource. [4]

   The amount of data which is generated is huge and it makes the processing of Big Data a difficult task. As discussed earlier previously only employees were engaged in data generation but by emergence of social networks and smart phones every ordinary users of computer and certain devices started to generate data.

2. **Velocity:** This is about the rate of increase (generation) in big data. Since there are many devices (sources) that can generate data such as phone, tablets, PCs, cameras , and etc; and the number of them is increasing as well this rate is continuously increasing. Because it is continuously increasing, we will never have a situation of analyzing static fixed data. It is never current or up-to-date.

   Due to participation of people in data generation the speed of data generation as a consequence has increased and we started to generate data in an enormous speed. So we needed some solution that be able to perform fast

[2]https://about.twitter.com/company

I/O without that much of latency.

3. **Variety:** The variation of data is another challenge of Big Data monitoring and management. Different types of data are: pictures, videos, spreadsheets, and etc. variety represents the type of data that is stored, analyzed and used. The type of data stored and analyzed varies and it can consist of location coordinates, video files, data sent from browsers, simulations etc.

   The challenge is how to sort all this data so it can be ”readable” by all users that access it and does not create ambiguous results.[4]

The aboves were the most famous challenges of Big Data which are known as 3V's of Big Data. However with more understandings and efforts regarding Big Data, some claim there are more V's of Big Data such as ”Value, Veracity, Validity, Volatility, etc” but we stop in these three most known characteristics in this project.

### 2.1.4 Why do we need Big Data?

Big Data means capturing almost everything. Companies can find all their necessary information in Big Data to better serve their customers.

For example Facebook Inc, is recording all information its users share in their Facebook profile. They keep track of all of its users' activities from liking a page, to commenting on political article and even the mood and feeling of them. It is basically capturing behavior of people in order to find their interests and needs and use those information to target their advertisements more relevant.

Data scientists, high-ranking professional with the training and curiosity to make discoveries in the world of Big Data, are able to generate analysis and statistics out of Big Data which can help managers in making a better decisions.

One of the most useful application of Big Data is collaborative filtering. When you purchase or search for some items on some online they keep track of your search and lookup for all other customers who searched for that item and get some common elements that they bought or looked for and return those as a suggestion to you.

As you can see we are able to get a lot of usage from Big Data and it still has many hidden sights that need to be discovered. It is a good time to be in field of Big Data and in the near future there will be more Data Scientist than any other scientist and there is a claim that most of other scientists will be merging to data scientist.

In the next section to as support of our discussion of how to processing and get use of Big Data, I am going to explain about Hadoop. A platform that has helped a lot in processing and solving issues of Big Data.

## 2.2 Hadoop

In order to get the most out of Big Data we need the right tools to be able to deal with all the challenges of Big Data.

Hadoop project includes these modules:

1. Hadoop Common: The common utilities that support the other Hadoop modules.
2. Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
3. Hadoop YARN: A framework for job scheduling and cluster resource management.
4. Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

Hadoop is a solution that uses cluster of commodity machines to perform Big Data storing and processing. Its two main component are HDFS (Hadoop Distributed File System) and MapReduce.

HDFS is self-healing, high-bandwidth cluster storage. If we were to store one terabyte of data in our HDFS cluster, it would break it up into storage blocks and then distributed it across all the nodes in our cluster.

MapReduce on the other hand is processing and data storage part of Hadoop. It is a Java-based programming module with two major functions (Map and Reduce).

We will talk in detail about HDFS and MapReduce in their own sections.

### 2.2.1 Why Hadoop?

Hadoop is *Scalable*, *Flexible*, *Fault-tolerant*, *Intelligent*, and *Open Source.*

#### 2.2.1.1 Hadoop is Scalable!

Scalability factor of Hadoop is inherited from its distributiveness. Since we are splitting the data into blocks and spreading them across our cluster adding more nodes will be easy, and this adding additional nodes not only does not disturb any process and storage but also will increase our computing power.

#### 2.2.1.2 Hadoop is Flexible!

Although MapReduce jobs are Java-based application; Hadoop allows us to adds non Java applications in order to be able to easily retrieve data out of clusters.

For example Facebook which is one the major users of Hadoop and has developed a project named Hive which is allows programmers with SQL knowledge be able to retrieve data out of their Hadoop clusters without having knowledge of Java and worrying about writing Java functions.[3]

Hadoop is also flexible in a manner that we can fit any data inside Hadoop. As long as we split up the data into storage blocks it does not matter that data is structured, semi-structured, or non-structured.

[3]Pig is another example of this kind of projects in Hadoop ecosystem which is developed by Yahoo Inc.

#### 2.2.1.3 Hadoop is Fault-Tolerant!

HDFS use a method called replication that gives Hadoop a very fault-tolerance power.

The idea behind replication is when HDFS want to break up the data into storage block there is replication factor which by default is three, that means there will be three copies of each block storage, and each of which will be stored in different locations.

When there is need for data retrieval and if one of the nodes that store that block of data is down HDFS will know that there will be copies of that missing block somewhere in our cluster and it will take them as replacement of the missing block.

In this way we will never lose any data or at least the chance of losing data will decrease.

#### 2.2.1.4 Hadoop is Intelligent!

Hadoop is bringing computation to data and not data to computation.

MapReduce and HDFS are integrated tightly in Hadoop to make sure MapReduce task run directly on the HDFS nodes that hold the required data.

On the top of that on a multi-rack Hadoop tries to keep block of data as local as possible and near together.

#### 2.2.1.5 Hadoop is Open Source!

The last element that interests us and it is my point of view, which is less focused in literature that We reviewed and it is also a motivation that We considered when choosing this topic, is that Hadoop is open source.

This means every programmer can contribute to its improvement and can add more and more functionality to it. This is the beauty of open source applications which enables the project to always improve. It allows more and more people start using Hadoop and make Hadoop a perfect ecosystem for Big Data processing.

### 2.2.2 HDFS & MapReduce

Hadoop uses master/slave architecture for both distributed storage (HDFS) and distributed computing (MapReduce).

#### 2.2.2.1 HDFS

It is a file system which is able to store large number of data sets in cluster of commodity machines. HDFS architecture is designed in such a way that data can be scaled out across clusters of nodes.

In the early days of storage, data was stored in one large capacity machine and if the size of that data needed to grow, the technicals had to buy a machine with larger capacity and copy everything in the new machine. This method was called *scale up* and would cost too much, for example if you were about to buy a machine with the capacity equivalence to four separate PCs, it would cost more than buying those four PCs.[5]

It was one of the advantages of HDFS over the old method of scaling up the machine, that it was able to store pieces of Big Data among multiple of machines and it reduced cost of buying new big servers.

#### 2.2.2.2 Advantages of HDFS

1. **Easy to scale out** As we mentioned earlier HDFS splits up large amount of data into small pieces of storage blocks, thus it does not matter how big or what type of data we are storing because at the end of the day all of them will be split up into same storage blocks and all of them will be treated as they are the same.

   Hence, these blocks can be stored in different locations and as soon as we run out of storage in one machine we can add up another one and pass out theses blocks to that new machine, this method of storing data gives the ability of easy scale out to HDFS over the other distributed files systems in market.

2. **Replication Factor** Replication Factor is the number of copies that HDFS make from each storage block. In order to avoid data loss upon node or rack failure HDFS stores these copies of the same block into different nodes, or in multi-rack environment, into different racks.

Hadoop Distributed File System has two main components: NameNode and DataNode which are basically two different software to manage the file distribution in Hadoop environment.

### Components of HDFS

**NameNode** is basically the master of HDFS and it is in charge of file system namespace management operations and block mapping.

When the application want to perform a task such as creating or deleting a directory, storing file and all other tasks, they should go through NameNode. NameNode decide which DataNodes to perform that application request it will start communicating directly to those DataNodes.

If the HDFS is running on multi-rack cluster as an effect of Rack Awareness NameNode will spread some of the data blocks into other racks withing the same network in order to save those portions from situations of Rack failure. For example if the replication factor is three one of the block will be saved in DataNodes on the other racks and two block will be in the same rack.

**DataNodes** preform all instructions they get from NameNode. Once the NameNode decided which DataNodes should perform the tasks the applications directly talk with DataNodes. Once the DataNodes get the instruction from NameNode and get the replication factor it replicate the data blocks and spread them across different nodes and then send the block id and the location of successful storage back to NameNode.

DataNodes that are up and running are supposed to send a continuous heartbeats to NameNode in order for the NameNode to understand that those DataNode are alive. If the NameNode does not receive heartbeats from a DataNode it consider it dead and the next time application needs those dead block the NameNode will direct the application to the DataNodes that have copies of those missing blocks.

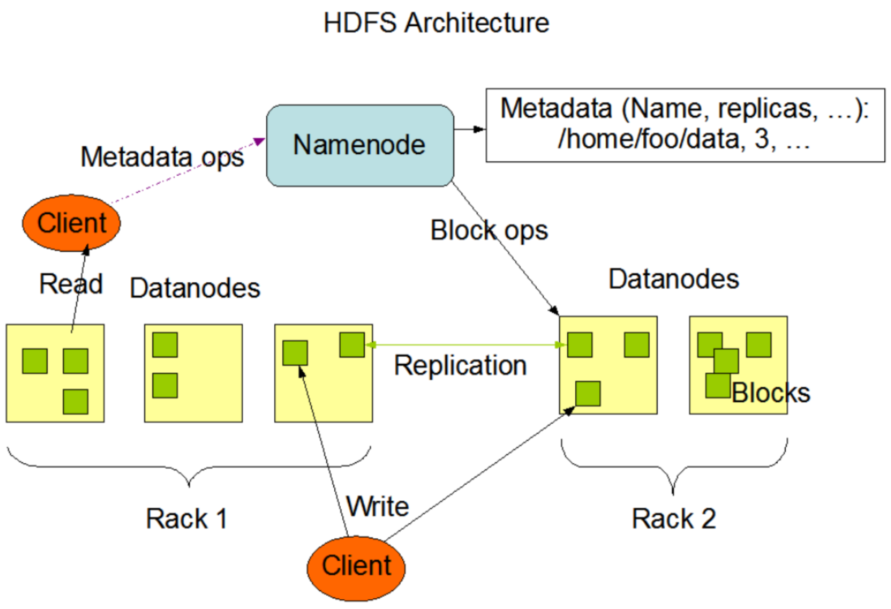


FIGURE 2.2: HDFS Architecture

#### 2.2.2.3 MapReduce

MapReduce is a programming model first used by Google Inc. to process and generate large amount of data which can be run on commodity machine. A typical MapReduce computation processes terabytes of data on thousands of machines. MapReduce computations conceptually are fairly easy to understand but the amount of data to be processed is huge and the focus factor is to reduce the time of processing. [6]

In order to reduce the time of processing the MapReduce can be executed in parallel and on large number of commodity machines. Google hid details of parallelizations, fault-tolerance, data distribution, and load balancing in a library and designed an abstract module of MapReduce , thus a programmer without a knowledge of parallel processing can also run MapReduce applications against cluster of computers.[6]

This abstraction was inspired by Map and Reduce function presented in Lisp and other functional programming languages.

A MapReduce job splits a large data set into independent chunks and organizes them into ¡key, value¿ pairs for parallel processing. This parallel processing (i.e. running independently on the DataNodes) improves the speed and reliability of the cluster, returning solutions more quickly and with more reliability. Typically both the input and the output of the MapReduce job are stored in a file-system.

The Hadoop framework (YARN, NameNode, DataNode) takes care of scheduling tasks (i.e. programs to execute), monitoring them and re-executes the failed tasks.

- The Map function divides the input ¡key, value¿ pairs into ranges and creates a map task for each range in the input. The NameNode distributes those tasks to the worker nodes (DataNodes). The "program that is supposed to run" then executes against the data on the DataNodes (Remember - bring the program to the data). The output of the program's execution is again attached to the ¡same key, but new value¿ pair. Note: the "key" remain the same, and the "new value" is the result for the program run for each corresponding key. Next, all of the sets of outputs of each program run is partitioned into a group of ¡key, value¿ pairs to be processed by the Reduce function.
- The Reduce function then collects the various results (i.e. ¡key, value¿ pairs) and combines them to answer the larger problem the master node was trying to solve. Each reduce pulls the relevant partition from the machines where the maps executed, then writes its output back into HDFS. Thus, the reduce is able to collect the data from all of the maps for the keys it is responsible for and combine them to solve the problem.

Figure 2.3 shows how MapReduce works.

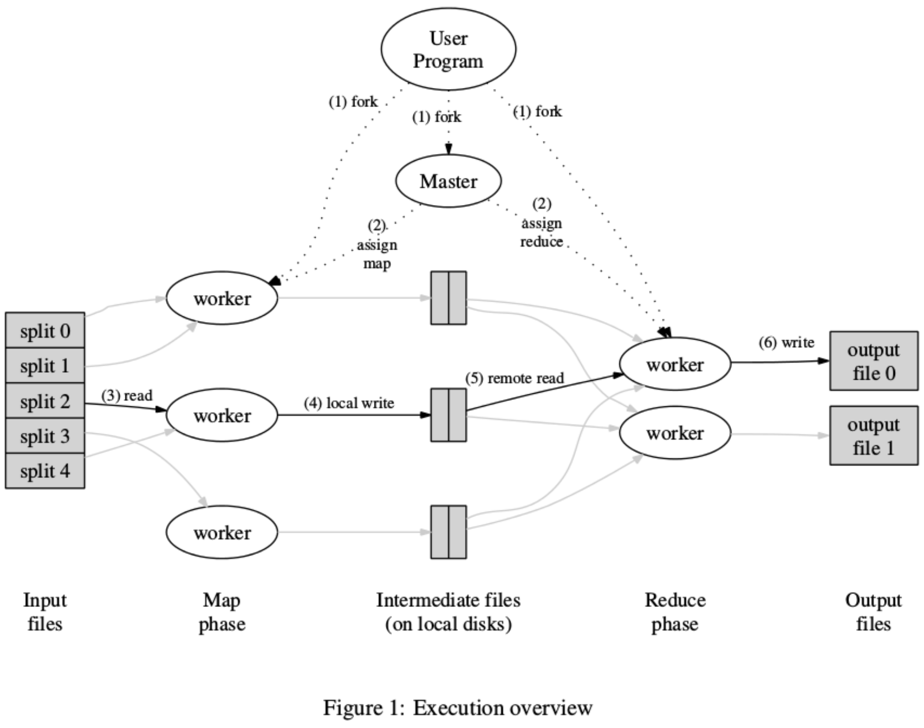


FIGURE 2.3: MapReduce Execution overview

The example of WordCount program that implements MapReduce can be seen in Appendix B.

### 2.2.3 Comparing SQL Database and Hadoop

Although we can create SQL-based application on top of Hadoop it is good to have a comparison between how Hadoop store data and how the SQL databases were storing the structured data.

1. SQL Database is for structured data but Hadoop is suitable for unstructured data as well as structured data. From this perspective Hadoop provides a more general paradigm than SQL.
2. As the size of our database grows we need more space. To run bigger database you should run bigger machine. it will be costly to buy machines with more storage and power. Hadoop use commodity machine general purpose and cheap machines. You can't run a SQL database across multiple machines but Hadoop is made for running on multiple cluster of machines.
3. Key/Value pairs instead of relational tables.
4. Functional programming instead of declarative query: in MapReduce you specify steps in processing data but when you run a SQL query you state the result that you need and it is the database engine that decides how to derive those results. In MapReduce we have scripts and code but in SQL world we have queries.

   MapReduce allows you to process data in more general fashion than SQL.[1]

## 2.3 Drawbacks of Hadoop

The architecture choices made within Hadoop enable it to be the flexible and scalable data processing platform it is today.

1. But, as with most architecture or design choices, there are consequences that must be understood. Primary amongst these, is the fact that Hadoop is a batch processing system.

When you execute a job across a large data set, the framework will churn away until the final results are ready.

This means we can not use the system until all processes are done.

2. With a large cluster, answers across even huge data sets can be generated relatively quickly, but the fact remains that the answers are not generated fast enough to service impatient users. Consequently, Hadoop alone is not well suited to low-latency queries such as those received on a website, a real-time system, or a similar problem domain.

3. When Hadoop is running jobs on large data sets, the overhead of setting up the job, determining which tasks are run on each node, and all the other housekeeping activities that are required is a trivial part of the overall execution time. But, for jobs on small data sets, there is an execution overhead that means even simple MapReduce jobs may take a minimum of 10 seconds.[1]

4. MapReduce model is not general and can not be applied to every problem, such as machine learning problems, streaming , etc. we needed more general and faster approach than MapReduce to process Big Data.

5. MapReduce model is not easy to understand and be applied by new and amateur users so we need an easy to use and powerful platform that provides us different need of programming. We can not expect everybody start building different programs for same machine learning algorithms.

## 2.4 Summary

In this chapter we understood what Big Data is, and get to know with the power of Big Data and why Big Data is becoming more available as well as more understandable to us.

Then we talked about 3V's of Big Data and claimed that Hadoop is a good solution when we look at our old relational databases. But with the challenges of data and with the pitfalls of Hadoop we got satisfied that in order to race with speed of Big Data we need a faster horse (Spark) and Hadoop is quite slow.

Thus we need to move to a faster platform called Spark which its primary goal is fast processing of data.

At the very end we convinced ourselves that computer need to think like human and we need to give more space to them as they are faster and more accurate than men, which is the reason of us to choose Machine Learning Library as platform of our choice to drive problems of Big Data.

In the part two we will investigate Spark and MLlib as solution to the problems identified with Hadoop and MapReduce Model. We will explain what spark actually is and why it is a better than Hadoop MapReduce. We further study the Machine Learning Library (MLlib) of Spark and see what algorithms it covers and what type of machine learning problems it solves.

Our focus will be on classification algorithms of MLlib so in the part two we explain some of the most famous classification algorithms to better understand what different functions of Spark API (codes) handles which part of classification.

# Part Two

## 2.5 Apache Spark

Apache Spark is an open-source cluster computing framework originally developed in the AMPLab at UC Berkeley. In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's in-memory primitives provide performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well suited to machine learning algorithms. Spark can interface with a wide variety of file or storage systems, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, or Amazon S3.[7]

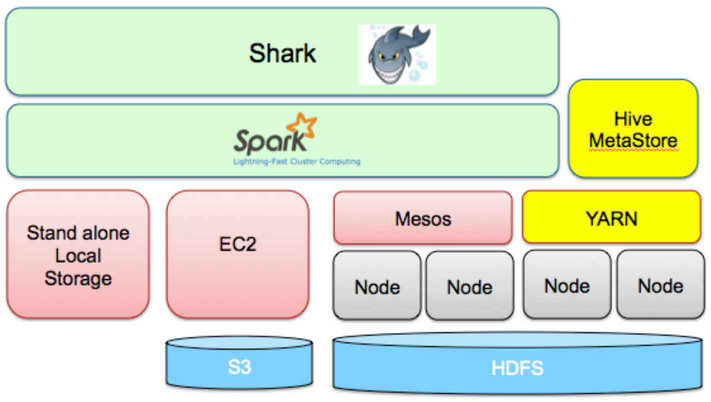


FIGURE 2.4: Spark Relation with Other Storage Systems

Spark is one of the most actively developed open source projects. It has over 465 contributors in 2014, making it the most active project in the Apache Software Foundation and among Big Data open source projects. Figure 2.4 shows how spark can be integrated with different technologies to enhance the Big Data processing.

Apache Spark is an open source cluster computing system that aims to make data analytics fast - both fast to run and fast to write. To run programs faster, Spark offers a general execution model that can optimize arbitrary operator graphs, and supports in-memory computing, which lets it query data faster than disk-based engines like Hadoop.[8]

Apache Spark is a fast and general-purpose cluster computing system which provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs.

It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

### 2.5.1 Why Spark?

One of the disadvantages of Hadoop is it provides data processing with latency in process. Data processing speed is limited to the speed of physical drives that data is stored in. In order to overcome this issue we needed to some system that can process data as fast as possible.

Spark implements in-memory processing rather than hard disk to hard disk process therefore it is faster. Spark idea came about in UC Berkeley AMPLab to solve this issues of Hadoop by Matei Zaharia in 2009. It was first open sourced in 2010 (first release), and donated to Apache Software Foundation on 2013 and Spark 1.0 was released on May 2014.
As you can see the age of Spark is not over 5 and it gained a lot of attention which itself is an indicator of its power and incredibility.

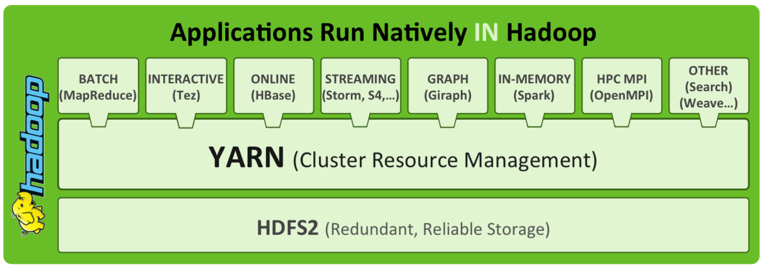

Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

FIGURE 2.5: Spark From Hadoop Point of View

As Figure 2.6 shows Spark gives a set of powerful tools that enable us to tackle different problems on large scale such as performing SQL queries, doing machine learning , graph processing and streaming and we choose Spark because of it can integrate easily with Hadoop as it is shown in Figure 2.5.
As you can see Spark can be used like any other Hadoop based application through talking with Yarn which is resource manager in Hadoop.

### 2.5.2 Spark Philosophy

In data mining world it was always a problem of scaling a good model to large scale dataset and many programmers spent alot of time with fighting with confusing and broken and limited API's. So Apache Spark comes with the philosophy of making life easy and productive for data scientists by doing the followings elements:

- Well Documented, Expressive API's: So developers and data scientist spend less time confusing in what different functions do. Being able to write Spark application in three different languages(Java, Scala, Python) can invite wide range of programmers with different backgrounds into Spark.
- Powerful domain-specific libraries: Covering four different areas (SQL, Streaming, MLlib, GraphX) make Spark one stop center for large-scale data processing.

- Easy Integration with storage Systems: Being able to get data in/out directly from HDFS, Cassandra, HBase, is one the main factors that attract many people towards Spark. And the ability to cache data in memories across cluster made substantial fast processing possible.
- Regular maintenance release: New releases does not destroy the tasks or codes written in older version but with little enhancement developers can get their old spark application working on newer versions of Spark.

### 2.5.3 Spark Architecture

As the figure 2.6 shows spark has unified stack that enables it to put multiple component closely together.At its core, Spark is a "computational engine" that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a computing cluster.

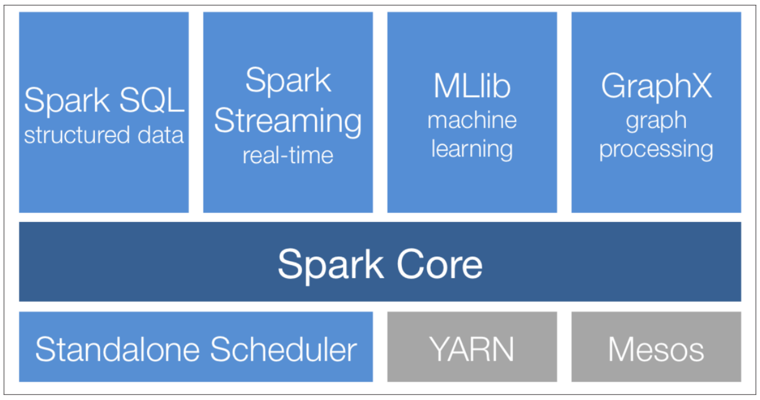


FIGURE 2.6: Spark Stack

This tightly integration has several benefits:

- First all libraries and higher level components in the stack benefit from improvements at the lower layers.

- Second, the costs associated with running the stack are minimized, because instead of running 5-10 independent software systems, we only need to run one.
- Third, each time a new component is added to the Spark stack, everyone that uses Spark will immediately be able to try this new component. Cost of downloading, deploying and learning changes to upgrading.
- Finally, one of the largest advantages of tight integration is the ability to build applications that seamlessly combine different processing models. e.g: we can write an application that can do machine learning on streaming posts (such as tweets in twitter) that are real time.

### 2.5.4 What is MLlib?

In order to understand what is Machine Learning Library we need to get to know what is machine learning at first. With machine learning we can gain insight from a dataset; we're going to ask the computer to make some sense from data.[9]

Some example of Machine Learning applications are, detecting Spam email, search suggestions, friend suggestion on social networks.

MLlib is a Spark implementation of some common machine learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction.[8]

### 2.5.5 Why MLlib?

With increase of Big Data, we started to derive more meaning out of data but human and his statistical is not going to solve much of a problems if he doesn't get help of computers.

Furthermore we can't repeatedly repeat easy tasks that are common for different areas of work, so we need to use computers in such a way that these machines get to learn and retrieve some meaning like humans does.

Hence we needed to be able to do machine learning on Big Data in an efficient way and Apache Spark has built-in machine learning library shortly known as MLlib which gives the following reasons to use:[10]

1. Ease of Use: MLlib can be used in Java, Scala, and Python. We can use any Hadoop data source(HDFS, HBase, or local files) which make it easy for us to use it in Hadoop cluster in GridLab.
2. Performance: MLlib consists of high quality algorithm which is 100 times faster than a MapReduce.
3. Easy to Deploy: MLlib can runs on existing Hadoop clusters and data.

## 2.6 Machine Learning

In all but the most trivial cases, insight or knowledge you're trying to get out of the raw data that won't be obvious from looking at the data.

For example, in detecting Spam email, looking for the occurrence of a single word may not be very helpful. But looking at the occurrence of certain words used together, combined with the length of the email and other factors, you could get a much clearer picture of whether the email is Spam or not. Machine learning is turning data into information.[9]

### 2.6.1 Who benefit from Machine Learning?

Machine learning lies at the intersection of computer science, engineering, and statistics and often appears in other disciplines. It can be applied to many fields from politics to Geo-sciences. It's a tool that can be applied to many problems. Any field that needs to interpret and act on data can benefit from machine learning techniques.

Companies are using machine learning to improve business decisions, increase productivity, detect disease, forecast weather, and do many more things. With the exponential growth of technology, we not only need better tools to understand the data we currently have, but we also need to prepare ourselves for the data we will have.[9]

### 2.6.2 Supervised Learning vs Unsupervised Learning

#### 2.6.2.1 Supervised Learning

Supervised learning is the machine learning task of inferring a function from labeled training data. The labeled training data (i.e. input-output pair) consist of a set of teacher training examples. In supervised learning, it basically means, the teacher will tell us that, if this is the input, then this is the correct output. The word "supervised" in English means to be directly in charge.

A supervised learning algorithm analyzes the labeled training data (i.e. input-output pairs) and produces an inferred function. The inferred function is then used for mapping new data, that is, given a new input, the inferred function will produce its predicted output. Supervised learning is learning by examples.
Classification is a type of supervised learning because the inferred function (i.e. classifier) produced by the teacher-trained learning algorithm, is used to predict which particular class any new input should go into.

**Classification:** Classification is checking whether an object belongs to a category (class) or not. In classification we define some attributes or characteristics for a category of things, animals, human race, disease, and etc. and if we can find these characteristics in other objects we simply put those objects in that category.

**Regression:** Regression is also a category of supervised machine learning that beside predicting a nominal value it can predict continuous values.

In regression we have set of data and our goal is to come up with a line equation that can pass through all data points or at least near to them in such a way that each point is at its closest possible distance of this line.

#### 2.6.2.2 Unsupervised Learning

Unsupervised learning is machine learning that is trying to find hidden structure (or information) in unlabeled data and to summarize and explain key features of the data. Labels are not given to the data and you try to extract information in general out of your data. In unsupervised learning, there is no label or target value given for the data.

In unsupervised learning, there is no desired output that is provided. Therefore categorization is done so that the algorithm differentiates correctly the different groups in the data (i.e. clustering of data). A task where we group similar items together is known as clustering. We run the algorithm on the data, separating the data into clusters based on some parameter. When we are finished, we analyse the clustered data and find useful pattern of information in it.

**Clustering** Clustering is basically is the task of grouping similar items together into different groups. We call each of these groups a cluster. Clustering is mostly done by k-means algorithm. It is called k-means because it is looking for **k** unique clusters, and each cluster has center which is the mean value of all elements in that cluster.

## 2.7 Labeled and Unlabeled Data in Machine Learning

### 2.7.1 Unlabeled Data

Typically, unlabeled data consists of samples of natural or human-created artifacts that you can obtain relatively easily from the world. Some examples of unlabeled data might include photos, audio recordings, videos, news articles, tweets, x-rays (if you were working on a medical application), etc. There is no ”explanation” for each piece of unlabeled data – it just contains the data, and nothing else.

### 2.7.2 Labeled Data

Labeled data typically takes a set of unlabeled data and augments each piece of that unlabeled data with some sort of meaningful ”tag,” ”label,” or ”class” that is somehow informative or desirable to know.

For example, labels for the above types of unlabeled data might be whether this photo contains a horse or a cow, which words were uttered in this audio recording, what type of action is being performed in this video, what the topic of this news article is, what the overall sentiment of this tweet is, whether the dot in this x-ray is a tumor, etc.

Labels for data are often obtained by asking humans to make judgments about a given piece of unlabeled data (e.g., ”Does this photo contain a horse or a cow?”) and are significantly more expensive to obtain than the raw unlabeled data.

### 2.7.3 Why bother about labels?

After obtaining a labeled dataset, machine learning models can be applied to the data so that new unlabeled data can be presented to the model and a likely label can be guessed or predicted for that piece of unlabeled data.

There are many active areas of research in machine learning that are aimed at integrating unlabeled and labeled data to build better and more accurate models of the world.

Semi-supervised learning attempts to combine unlabeled and labeled data (or, more generally, sets of unlabeled data where only some data points have labels) into integrated models. Deep neural networks and feature learning are areas of research that attempt to build models of the unlabeled data alone, and then apply information from the labels to the interesting parts of the models.

## 2.8 How to Run the Algorithm and Test?

Once we have decided about the algorithm we need to test it. The way it is done is to give the *training test* along with *target variable*

*Training test* is our known data set that we use to train our algorithm with known label points. *Target variable* is the expected output of our algorithm which is also known as class or label point.

After that we fed our *test set*, which is a set of uncategorized training example and the target variable is unknown, and we let the program decide what are the target variable for each sets.

Once we have our output, it will be tested or compared with the actual target value that it belongs to, hence we get some sense of how accurate the algorithm is. The number of wrong predictions over the total number entries of test set is known as *error rate*.

## 2.9 Study of Different Classification Algorithms

In this section we will explain our understanding of studying different classification algorithms in order to see which one is good for what kind of problem and based on this understanding we will choose one run and dig more on the our Hadoop cluster.

### 2.9.1 Classification Algorithms

1. **kNN (k-Nearest Neighbors):**
   In kNN algorithm we will have a training dataset with known labels for each entry in that dataset. When we are given a new data and asked to label it based on the training dataset, we will the distance between the features of that entry and features of instances of training dataset.

   Then we sort the distances with their corresponding labels in ascending order.

   We look at the top k most similar pieces of data (rule of thumb is $k < sqrt(n)$,n is number of examples) from our known dataset; this is where the k comes from.Lastly, we take a majority vote from the k most similar pieces of data, and the majority is the new class we assign to the data we were asked to classify.

   **Example** To get a better understanding of kNN lets take the following example to show how to label an unlabeled element using kNN algorithm:

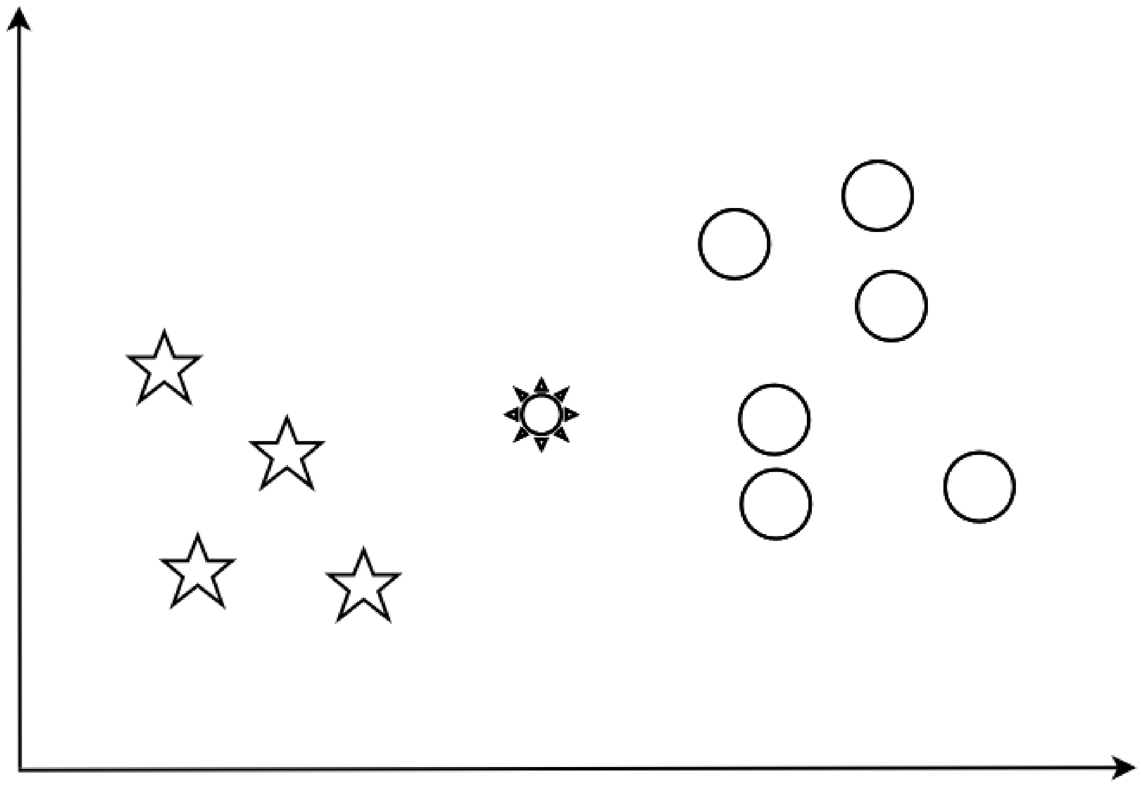


FIGURE 2.7: Shape classification problem

- In figure 2.7 we have 6 circles and 4 stars and lets assume we want to classify whether the shape in the middle fall into circle or start category.
- We can store coordination points of each shape in matrix form and find the distance of our unknown shape with its neighbors. So we represent starts as S where $S_1$ means star 1 and so on and we represent circles as C where $C_1$ means circle 1 and so forth and let our unknown shape

be represented as U

$$M = \begin{bmatrix} S_1 & x & y \\ S_2 & x & y \\ S_3 & x & y \\ S_4 & x & y \\ C_1 & x & y \\ C_2 & x & y \\ . & & \\ . & & \\ C_6 & x & y \\ U & x & y \end{bmatrix} \quad Label = \begin{bmatrix} Star \\ Star \\ Star \\ Star \\ Circle \\ Circle \\ . \\ . \\ Circle \\ ? \end{bmatrix}$$

- According to kNN algorithm we should find the distance of U with other elements and and sort them in increasing order. From our high school we know the formula for finding the distance between two point is as below:

  distance between A and B = $\sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$

- So we start calculating the distance between U and other points and then we sort them in ascending order.

- Now let's assume we sort the distances and classes with lowest distance with U are as follow and for the ease of understanding we put their

Labels along with them:

$$\text{Sorted based on the distance} = \begin{bmatrix} S_1 \\ C_2 \\ C_3 \\ C_4 \\ S_2 \\ C_1 \\ S_3 \\ S_4 \\ C_5 \\ C_6 \end{bmatrix} \quad Label = \begin{bmatrix} Star \\ Circle \\ Circle \\ Circle \\ Star \\ Circle \\ Star \\ Star \\ Cirlce \\ Circle \end{bmatrix}$$

- At this stage we determine k=3 and return the label of majority vote among the first 3 elements of our distance vector as our predicted label for U which in this case is "Circle" because we get 2 Circles out of first 3 labels.

2. **Decision Trees:**

   The decision tree is one of the most commonly used classification techniques; recent surveys claim that it's the most commonly used technique. [9]

   A *decision tree* is a tree that lead us toward our target value in hierarchical manner. After passing each level we will be closer to our target as we are putting more and more constraints on our target class.

   A decision tree has *decision blocks* (rectangles) and *terminating blocks* (ovals), where some conclusion has been reached. The right and left arrows coming out of the decision blocks are known as *branches*, and they can lead to other decision blocks or to a terminating block.

**Information Gain:** To build a decision tree, we need to understand a concept called *information gain*, which is the measure of change in information before and after the splitting data.

In order to calculate the measure of information we should calculate the value of each entity and sum them together to do this we use a formula called *entropy* which is "the expected value of information".
The information for symbol $x_i$ is defined as
$l(x_i) = \log_2 p(x_i)$

where p($x_i$) is the probability of choosing this class.

To calculate entropy, you need the expected value of all the information of all possible values of our class. This is given by
$H = -\Sigma_{i=1}^{n} p(x_i) \log_2 p(x_i)$

where n is the number of classes.

**Splitting The Dataset** We use entropy to find the best feature in our dataset to split on. Once we found the best feature to split we will split the dataset and all records will be traversed to the lower branches. If the lower level of our tree still has decision block we ran our algorithm to find the best feature on the decision block and split it recursively until we reach a terminating block. The class that could reach the end leaves of our tree will be classified same as the ending leaf.[9]

**When to stop splitting?** There are three stopping condition in decision tree:

(a) All classes in that split have the same label, so we return that label as our prediction;

(b) We ran out of features;

(c) we have no feature to split and our labels are different in this case we sort the data and the majority vote is our prediction for that node.[9]

**Building The Decision Tree** Once we know what the best feature and condition to stop the split are it is time to start building a decision tree.

We first start with whole dataset as first decision block(root) of our tree, second we find the best feature to split and split our dataset to decision blocks and terminating blocks. Once we know what data should go to decision blocks and what data should go into terminating block we transform them into the lower levels.

Now we need to go to our new decision blocks and repeat the same process until we have all of our dataset inform of terminating blocks.

**Classification with Decision Tree** To use our tree to classify some instance we should write a function to descent our tree until it reaches the lowest possible terminating blocks.

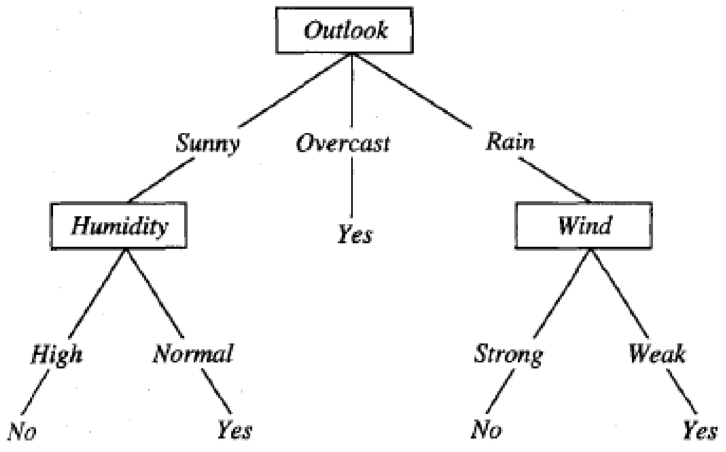


FIGURE 2.8: Classification Using Decision Trees

3. **Naive Bayes:**

   Naive Bayes (NB) is based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. It is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods.

   Bayesian decision theory in a nutshell: making a decision with the highest probability.

   If $p1(x, y) > p2(x, y)$, the class is 1 (blue crosses in figure 2.9).
   If $p2(x, y) > p1(x, y)$, the class is 2 (dark circles in figure 2.9).
   What the above mean is that we have an equation p1(x,y) for class 1(e.g: The stars) and we have another equation p2(x,y) for class 2(e.g: The circles) and if we are given a shape with new feature and asked to identify its class.

We compute p1 and p2 and if p1 is greater than p2 it means that these new data belong to class 1 or it belongs to class 2 if p2 is greater.

Naive Bayes is an extension of the Bayesian classifier and it is a popular algorithm for the document classification problem.[9]

Bayesian classifier is based on *conditional probability*. In probability theory, a *conditional probability* measures the probability of an event given that (by assumption, presumption, assertion or evidence) another event has occurred. [4]

**Conditional Probability in Classification** If we know the probability of some feature given some class using Bayes' Theorem we can calculate the probability of that class given features based on following formula:
$P(c_i|x,y) = \frac{P(x,y|c_i) \times P(c_i)}{P(x,y)}$

Using this formala we can form our concept of Naive Bayes classifier as below:
If $P(c_1|x,y) > P(c_2|x,y)$, the class is $C_1$(blue crosses in figure 2.9).

If $P(c_2|x,y) > P(c_1|x,y)$, the class is $C_2$(dark circles in figure 2.9).

[4]Wikipedia definition of Conditional Probability.

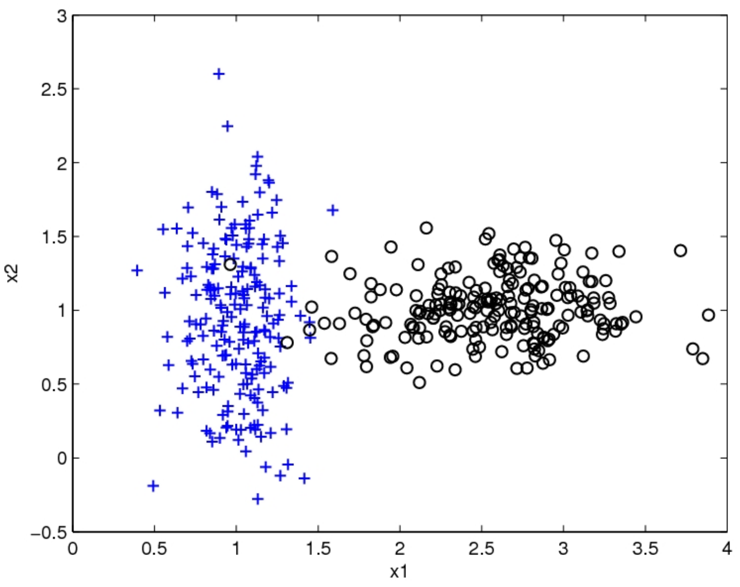


FIGURE 2.9: Classification Using Naive Bayes

**Naive Bayes in Action** In order to use Naive Bayes algorithm we need to calculate the probability of all features in our **training dataset** based on the label (class) those feature belong to.

Once we have the probability we can use Bayes theorem to find the probability of class given those features in our **test dataset**.

4. **SVM (Support Vector Machine):**

   In the SVM algorithm we try to separate our N-dimension data set with N-1 dimension *hyperplane*, the wall between different classes in our data set.

   For example if we have a dataset that can be plot in 2-D co-ordinate system as shown in figure 2.10 our hyperplane is a line , similarly if our dataset can be plot in 3-D space we need a plane to separate different data from each other.

The closest point in each data group and separating hyperplane is called *Support Vectors* and the distance of the support vector and separating hyperplane is know as *Margin*.

The goal of SVM is to maximize the margin and the reason of this maximization is that if we made a mistake or our trained example data was small, we'd want it to be as robust as possible. Based on figure 2.10 our best hyperplane is line $H_3$ which makes the maximum margin.

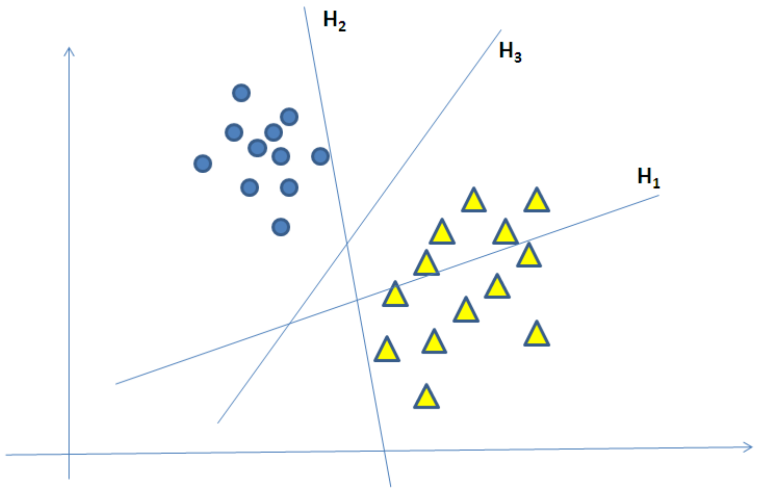


FIGURE 2.10: Classification Using SVM (Support Vector Machine)

5. **Logistic Regression:**

   In logistic regression we deal with optimization algorithms.Some examples of optimization from daily life are these: How do we get from point A to point B in the least amount of time? How do we make the most money doing the least amount of work? Things we can do with optimization are quite a lot in the coming paragraphs we talk about few optimization algorithms.

   Regression us is to find best-fit line for given set of data points. In logistic regression we have a bunch of data, with data we try to find a function for that set of data. Finding best fit is similar to regression, and this function is how we train our classifier.

   **Classification with logistic regression** Logistic regression have some advantages for example it is computationally inexpensive and it is easy to implement and knowledge that we get out of it is easy to understand, However it is prone to under-fitting, meaning there is less parameters than need for the model, logistic regression may have low accuracy[9].

   For the classification using logistic regression we would like to have equation that we give all of our features and it will predict the class.

   $z = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$

   In two-class case (which is also known as Binary classification), the function will spit out a 0 or 1 based on the feature we give to it.

   $\sigma(z) = \frac{1}{1+e^{-z}}$

   **Optimization** Assuming the function $z = w_0x_0+w_1x_1+w_2x_2+...+w_nx_n$ is our classification function we need to optimize it to get best result out of it. By optimizing we mean that we should find the best coefficient **w** so this classifier will be as successful as possible. There are several optimization algorithms such as Gradient descent, Stochastic gradient descent, Gradient ascent, and etc.

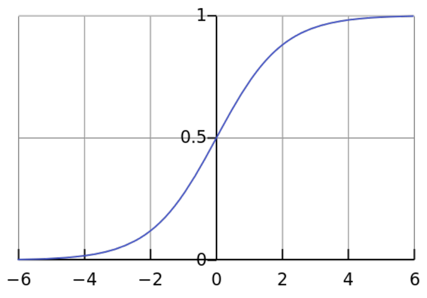


FIGURE 2.11: Binary Classification Using Logistic Regression

6. **AdaBoost:**

   Adaboost is one of the most famous meta-algorithms.Meta-algorithm is combining different algorithms to increase the performance of our classification. AdaBoost is considered by some to be the best-supervised learning algorithm.

## 2.10 Summary

In part two of literature review we talked about the need of using Apache Spark. We got to know how it started and understood that lack of unified software for data scientist made Apache Spark a trivial choice to do data mining.

Later we talked about the architecture of Spark and benefits of this type of architecture, such as; improvement of lower layers improve the higher level components as well, cost effectiveness of Spark, exchange between download cost with upgrade cost, and last and most important benefit combination of different processing model such as machine learning with streaming.

We also studied also studied about machine learning and different type of machine learning. By now we know what is the difference between supervised machine learning and unsupervised machine learning and we learned that classification is all about getting a machine to work by its own by predicting some label for an unlabeled set of entities based on previously labeled entities.

We then went deep to some the most used classification algorithms such as k-Nearest Neighbor, Decision Tree, Naive Bayes, Support Vector Machine, and Adaboost and understood which algorithm is used for what task.

# Chapter 3

# Methodology / Requirements

In this chapter we will explain the tools and methods that is needed to help us solve the problem that we have in hand.

## 3.1 Selection of Hadoop

There are many infrastructure and frameworks that can do big data processing, both commercial and open source. Examples of commercial are Cheetah, DataTorrent, MillWheel, and etc. Examples of open source are Apache Crunch, Apache Pig, Apache DataFu, and etc.

We decide to select Apache Hadoop because it is free, open source, so we can see the underlying codes. We can run and understand HDFS and MapReduce in Hadoop.

Since Big Data processing is a wide area of study and since we did not understand it very well at the beginning of project we decided to start from scratch and learn everything from the beginning.

For the phase one of this project we decided to first start to understand what Hadoop framework is, get the idea of how it work and handle Big Data by reading through online documentation that is available on Apache website.[1]

As a research project, we will learn how to install and configure different versions of Hadoop on both single node and multi-node environment. In this way we will practically see how the system works and we will get a full understanding of how to setup different part of Hadoop.

Once we know how configure and setup our cluster we will move to running simple examples of MapReduce application to see the basic steps in writing MapReduce application and get the main concept of writing such programs.

In the first phase of our project we put most of our focus on Hadoop and its two core component, HDFS and MapReduce, and we have not gone to Spark and Machine learning library and we have planned to focus more on it on the second phase of project. However we will look at how to install and configure Spark on our cluster to have an idea for the second phase.

[1]http://hadoop.apache.org

## 3.2 Selection of Spark

Although there are many infrastructure and frameworks that can do fast data processing, both commercial and open source we decided to select Spark as it covers wide area of fast data processing such as Streaming, SQL, Machine Learning, Graph Processing, which all together almost cover all problems and aspects of fast data processing of Big Data.

Another reason of choosing Spark was it's open source which as we mentioned earlier allow us to actually see the processes and codes. Along with this compatibility of Spark to extract data from HDFS the main distributed file system of Hadoop gave us more interest on going toward Spark.

Apache Spark API is available in three different languages of Java, Scala, and Python and this give us a sense of comparison between three different languages. we can see how different languages have been used to tackle the same problem and perform the same tasks which itself can be a good learning point.

Spark not only does machine learning but also provide powerful APIs for graph processing, SQL-like queries, and streaming and once we know how to deploy MLlib API we can get good sense of how to use other APIs as well and it can be really important in future studies.

## 3.3 Selection of Machine Learning Package

There are many infrastructure and frameworks that can do Machine Learning processing, both commercial and open source.
We decide to select Apache Spark and MLlib because it is also free, open source and so we can see the underlying codes.

In addition, Apache Spark and MLlib are packages that were designed to work with Apache Hadoop, so we want a guarantee that we do not have problems in implementation. Our goal is to understand how machine learning is implemented on a large scale running on top of the Hadoop infrastructure.

## 3.4 Proposed Problem to be Solved

Since we are doing the simulation of Big Data Processing and Machine Learning of large scale and due to machine limitations in Grid lab we do our machine learning on smaller dataset.

The proof of concept here is if we are able to do machine learning and specifically classification part of it on our found dataset it is proved that we can do the same method on Big Data and in large scale consequently.

For our problem we have a dataset called Adult[2] which has the following file:

1. adult.data: It is our dataset with 32561 instances and 14 features that need to be classified. The features have been separated by comma and each instances is on a newline.

[2]The dataset is Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: $((AAGE > 16)\&\&(AGI > 100)\&\&(AFNLWGT > 1)\&\&(HRSWK > 0))$

2. adult.test: It is our training example consists of 16281 instances and 15 features(including the earning of people).

3. adult.names: It is the updated dataset description file. It was modified 31-Jan-2001.

4. old.adult.names: It is the first dataset description file and it was written on 10-Aug-1996.

All of the above files will be found in the disk and not in appendix since it is very large data to print and can not be put in appendices.

Prediction task is to determine whether a person makes over 50K a year.

## 3.5 Selection of Algorithms

Our task based on the dataset is going to be a classification problem. We are going to classifying whether an individual is making greater or equal to $50,000 or less over the course of one year.

Basically we have two class of people, those who 1. make more than or equal to 50K or 2. less than 50 K, to classify them into two groups.

It is a classification problem because we are going to predict a yes/no value which can be express as 0 or 1 numbers which are discrete values.

## 3.6 Classification with Logistic Regression

For the task of classifying using Spark API we are going to use logistic regression which is an algorithm in Machine Learning for Classification. Classification involves looking at data and assigning a class (or a label) to it. Usually there are more than one classes, but in our example, we will be tackling Binary Classification, in which there at two classes: 0 or 1. Essentially what we do, is to draw

a 'line' through our data, and say If this data point (salary less than or equal 50K) falls on one side, assign it a label 0, or if it falls on the other side of the line(salary greater than 50K), give it a label 1. Machine Leaning 'fits' this line using a optimization algorithm (usually Gradient Descent or some form of it), such that the error of prediction is lowered.[11]

# Chapter 4

# Implementation Design

In this chapter we explain our design process that need to be implemented as prototype application that will help us to do run application on Hadoop and also do classification on Spark.

## 4.1 Design Overview of Running an Application in Hadoop

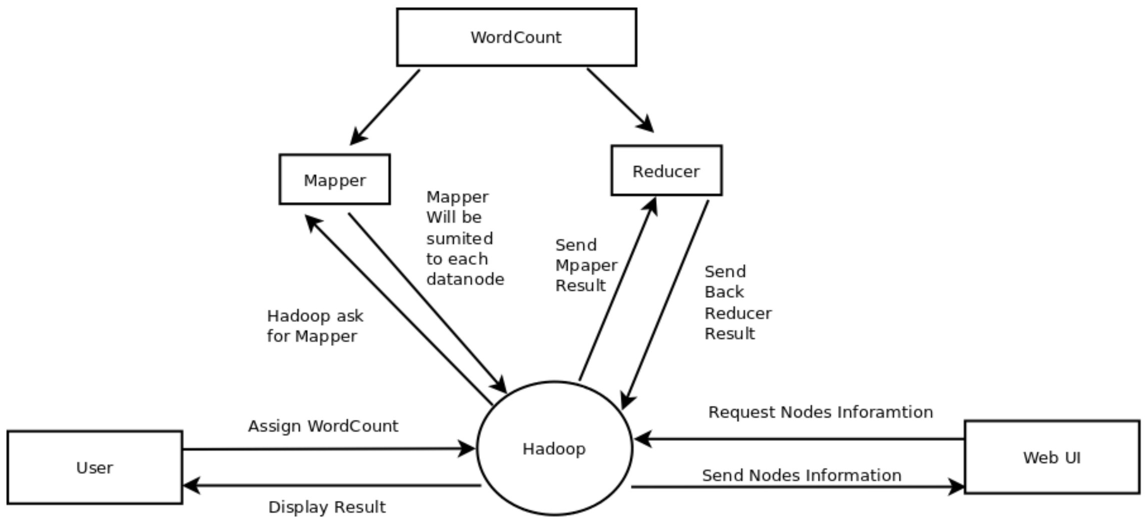


Figure 4.1: Overview of WordCount MapReduce on Hadoop

Based on our design in figure 4.1 we will write a WordCount program in java which is consist of two main subclass as it is required in MapReduce programming model. Once we have our program user will assign this application to Hadoop to run.

Once the application has been assigned to Hadoop, it will ask one Mapper for each DataNodes and will also send back the result of each Mapper which reside in local memory of each DataNodes to Reducer. At this point Reducer will combine all same key values in iteration manner and send back a list of same keys to Hadoop.

In order to see the result in an interface Hadoop provide a Web UI to see all data in more readable and intuitive way. All data stored in HDFS can be viewed using this Web UI.

## 4.2 Setup and Installation of Apache Hadoop

In our implementation design , we need to download the correct version of Apache Hadoop to get it working both on a local machine and on a cluster. We will have to test with a few latest version like Hadoop version 2.20, version 2.4.1 and version 2.5.0. We will be running them on the Linux Operating System.

As a research project, we will have to run Hadoop, see how it works and understand how the software works. We will provide the installation and run reports in Chapter 6.

To make sure the Hadoop infrastructure we installed is working properly, we will run a WordCount program. We must make sure that the results are as expected, for example checking against published results. We also will provide this report in Chapter 6.

## 4.3 Setup and Installation of Apache Spark and MLlib

After successfully installed Apache Hadoop, we need to install Apache Spark and MLlib packages. We will provide the report for the installation of Apache Spark and MLlib in Chapter 6.

After successful installation, we must check to make sure the Apache Spark and MLlib we installed are working properly. We will have to look into the MLlib examples to run for this check. We also will provide this report in Chapter 6.

In this chapter we will be explain how to install Hadoop on a single node machine as well as multi-node cluster of machines. The purpose of this chapter is to practically see how to install and configure Hadoop in our desired environment and understand how to set up Hadoop to get the most out our resources.

At the end of each section we will demonstrate simple a WordCount example to grasp the differences and similarities between two different environment.

WordCount program is equivalent of "HelloWorld" in the basic programming course which is mostly used by Big Data programmers to depicts how DFS and MapReduce work together, and its primary function is to count the number occurrence of words within a given text file.

## 4.4 Components for distributed execution in spark

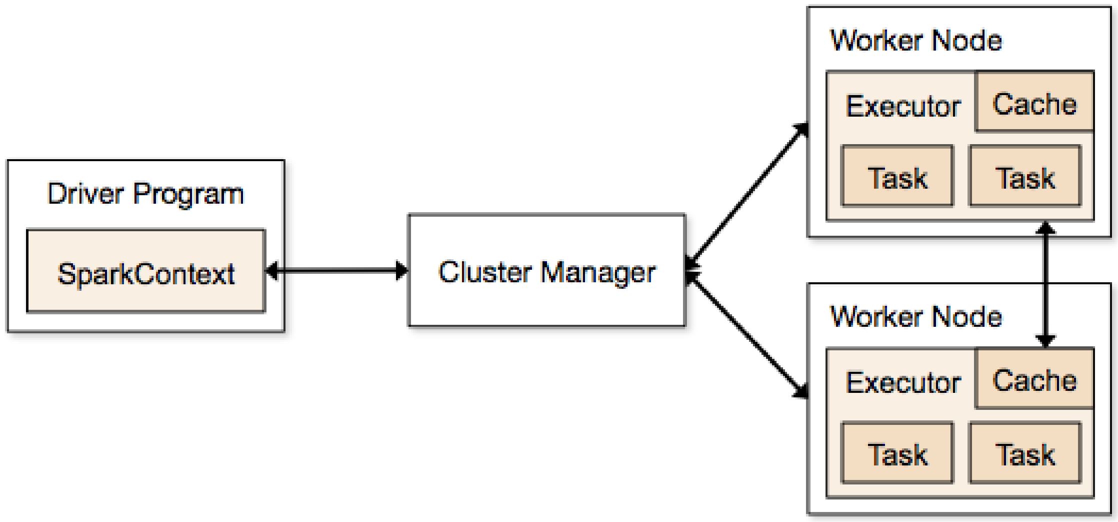


FIGURE 4.2: Components for distributed execution in Spark

As figure 4.2 shows Spark application consists of a driver program that launches various parallel operations on a cluster. *Driver* contains our application main function and defines a distributed datasets on the cluster, then applies operations to them. Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster.

*SparkContext* is our gate to cluster.Once we have SparkContext we can create resilient distributed datasets, or RDDs. An RDD is simply a distributed collection of elements that can be worked on them in parallel.

## 4.5 Adult Dataset

The dataset as we explained in earlier chapters is census dataset on different range of people from age $> 16$ and $< 100$. We will be predicting a person earns more thank 50K over a course of a year or less using decision tree and naive bayes algorithm on MLlib. This type of prediction is known as Binary Classification, since it is only two class to be identified and be predicted. Normally these two type is represented as -1 and +1.

## 4.6 Datasets Accounting (House Keeping Stuffs)

Preparing the dataset means we can not directly use the datasets we have downloaded from UCI repository because machine learning algorithms can not use any format of the data. To be able to calculate the error rate, misclassified items, percentage of correct outcome we need to do some house keeping stuffs such as data cleaning, marking, and et to prepare our dataset that best fits our experiment of classification.

In order to achieve the goal of dataset accounting we have ignored separation of adult.data (default training example from UCI repository) and adult.test (default test example from UCI repository).
Set of actions to be taken against dataset can be itemized as follow:

1. Combine adult.data and adult.test to get a unified initial raw data
2. Mark the records of initial raw dataset, meaning put the line number for each record so later when we predict we can always come back to the source and check if we get the correct prediction.
3. Split the raw dataset into two identical dataset of training and test examples based on some specific percentage say 50% train, 50% test.
4. Keep record of different classes before and after split

The above were set of tasks for dataset part we do have some more set of tasks for the overall algorithm steps which should be followed by these step to make classification system which will be covered in overall system design.

## 4.7 Binary Classification

As we studied in Literature Review chapter Binary classification is the task of classifying and object that can belong to two possible class. Examples of binary objects are an email can be Spam or not, breast cancer being malignant or benign, passing or failing a subject, and many more examples that we deal with it in our day to day life.

Binary classification aims to divide items into two categories: positive and negative and our dataset also have only two possible scenarios for each record; either earning more than 50,000 ($> 50K$, positive, 1) or less than or equal 50,000 ($<= 50K$, negative, 0), so our task will be designing a binary classification on this dataset.

MLlib supports two linear methods for binary classification: linear Support Vector Machines (SVMs) and logistic regression among which we will go with second option "logistic regression".

## 4.8 Binary Classifier Design

Logistic regression API that we are going to use needs each of the positive and negative classes to be in separate files prior to deploy and converting them into RDD.

We will solve this problem by writing a Java program that is able to do any customized separation on dataset.

In order to do the classification with logistic regression we need first prepare our dataset to do the following:

1. Start with an RDD of strings representing our dataset.

2. Run one of MLlib's feature extraction algorithms to convert text into numerical features (suitable for learning algorithms); this will give back an RDD of vectors.

3. Call a classification algorithm (e.g. logistic regression) on the RDD of vectors; this will give back a model object that can be used to classify new points.

4. Evaluate the model on a test dataset using one of MLlib's evaluation functions.

## 4.9 Analyze

Once we have our classifier we need to analyze the behavior of our classifier, this means we will test different partition of datasets and capture the outcome and compare them together and study it.

At this point we have reached the primary goal of this project which was to understand how machine learning works and get to know the basics of designing a machine learning system.

## 4.10 Summary: Overall System Design

This section is the summary and also a sense of overall system design of this project. As a result of being research based project system design is straight forward and simple because we are not going to make a very complicated application to perform some specific task but what we are more interested and should be is the insight we get out of application. This application is going to help us as tool of getting better understanding of the researched topic and draw some conclusion to grasp the concept of machine learning and classification on large-scale dataset and Big Data.

Hence we can put the overall flow in the following list:

- Hadoop installation on clusterrocks Grid lab in FCI, before of everything we need to learn and install Hadoop to be able to store our dataset on its distributed file system (HDFS) and process them.
- Spark installation on Hadoop cluster, it will give us the understanding of how we can get Spark to work with HDFS.
- Dataset accounting, we get to know structure of data, how many features, and how many of records belong to category positive and how many of them fall into negative
- Creating Java program that can partition datasets into different formats.
- Making binary classification engine to do the classification on Adult dataset.
- Analyze the result of classification engine draw conclusion based on the result.

# Chapter 5

# Implementation Plan

We need to have Hadoop set and running. So in this chapter we will depict how the internal structure of Hadoop work together.
Once we are done with Hadoop we see how Spark is related to Hadoop and can work as processing engine of Hadoop.
Using simple and powerful tools of Ubuntu terminal and with help of little Java program that we have written perform the dataset accounting design that we discussed in Design chapter.
We then get use of Spark Java API (`http://spark.apache.org/docs/latest/api/java/index.html`) and programming guide in Apache Spark web page (`http://spark.apache.org/docs/latest/`) we write a program that can do logistic regression binary classification on desired dataset.

## 5.1 Flowchart of Starting Hadoop

Figure 5.1 is the flowchart of how to get the Hadoop up and running on Single Node machine after it has been installed successfully. Once we get our Hadoop up and running we are ready to run our simple test program called WordCount against on our Hadoop to check everything is working smoothly.

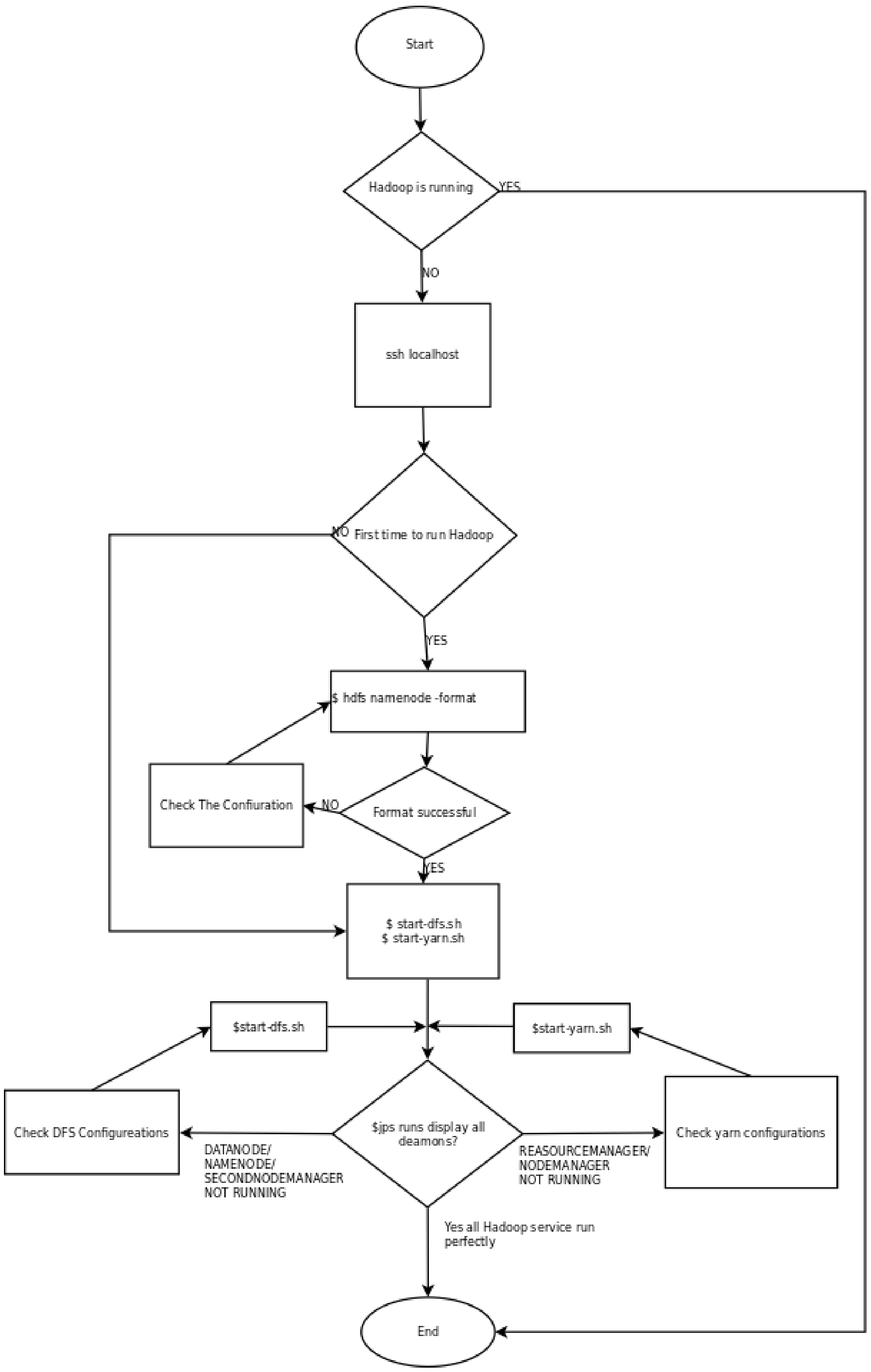


Figure 5.1: Flowchart of Running Hadoop

## 5.2 Flowchart of WordCount Example Program

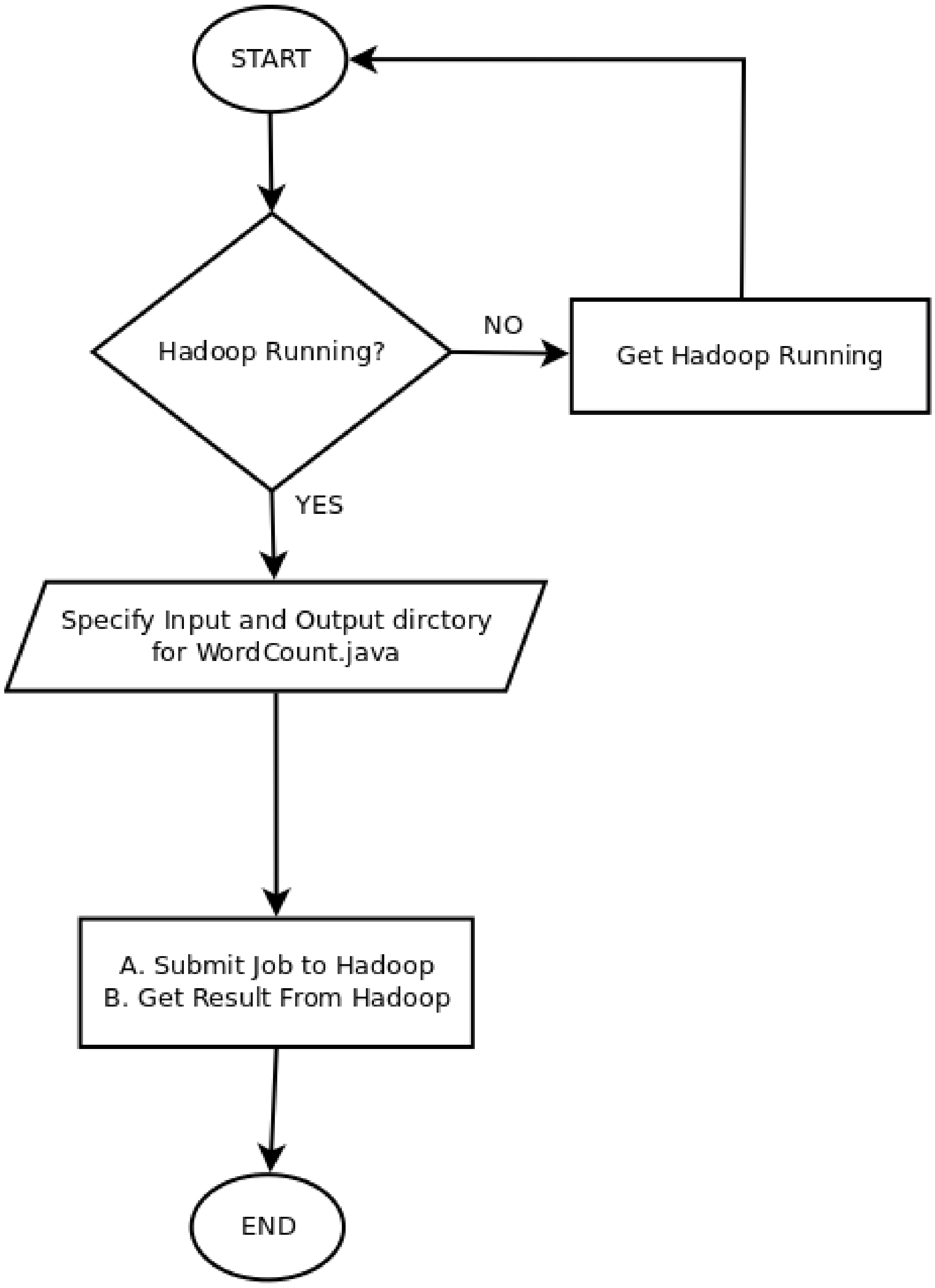

START
Hadoop Running?
NO
Get Hadoop Running
YES
Specify Input and Output dirctory for WordCount.java
A. Submit Job to Hadoop
B. Get Result From Hadoop
END

Figure 5.2: Flowchart of Running WordCount on Hadoop

Figure 5.2 shows how to submit our WordCount to Hadoop.

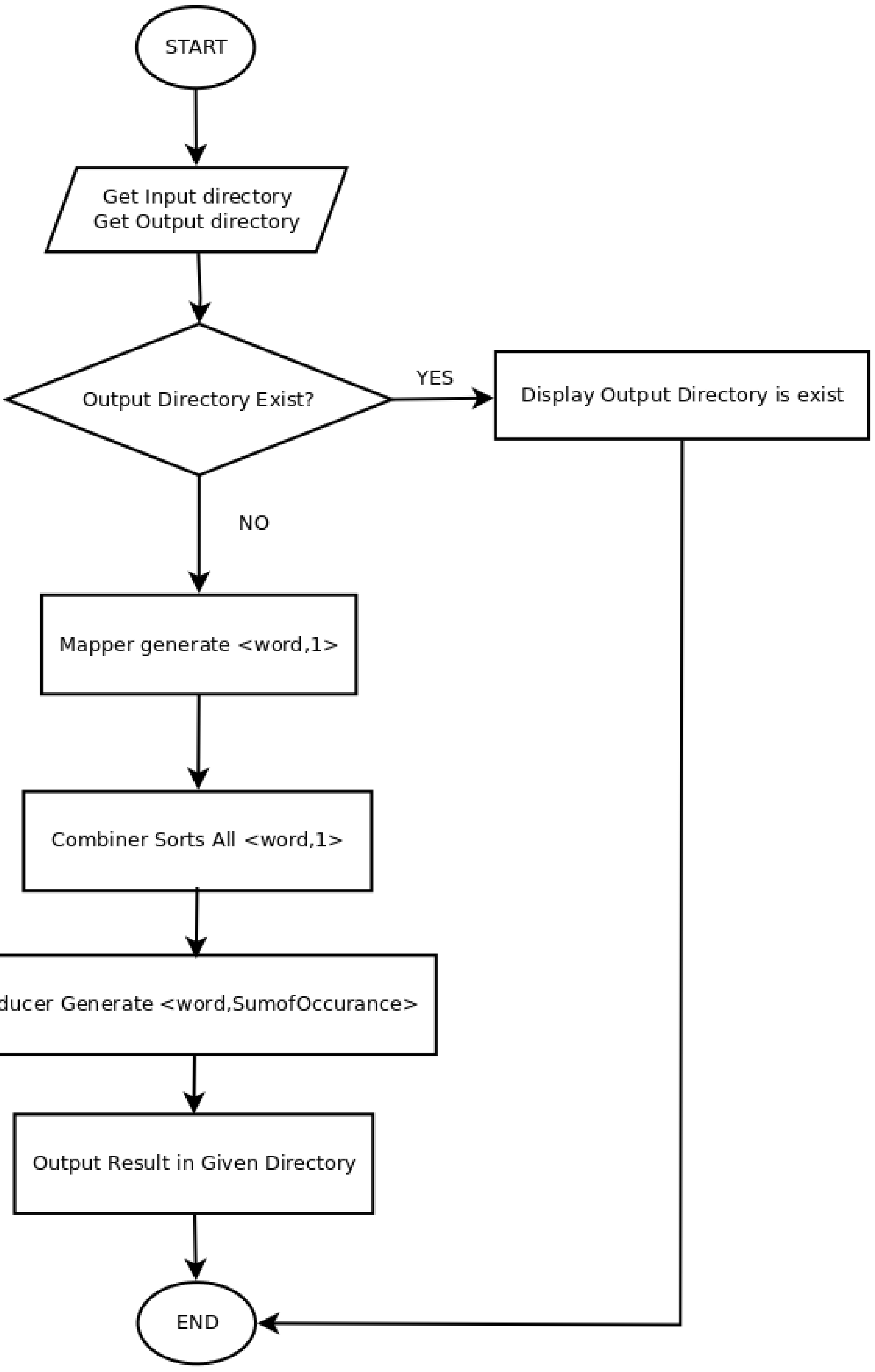

START
Get Input directory
Get Output directory
Output Directory Exist?
YES
Display Output Directory is exist
NO
Mapper generate <word,1>
Combiner Sorts All <word,1>
Reducer Generate <word,SumofOccurance>
Output Result in Given Directory
END

Figure 5.3: Flowchart of WordCount

Figure 5.3 is showing how WordCount program works and generate the number of occurrence of words from an input directory to an output directory. We will get use this illustration in our Phase two to scale up and generate more MapReduce jobs.

The actual source code is in Appendix B.
Based on the successful installations of Apache Hadoop, Apache Spark and MLlib packages, we will do the following:

1. Study the existing example applications that have been successfully implemented using MLlib on Apache Spark and Apache Hadoop.
2. Analyze the example applications that the uses MLlib package. Then choose one that can be done using the MLlib package.
3. Study the "big data" that was processed by the MLlib package on the selected topic.
4. Based on the selected topic, think of some similar problems that based on results of items (2) and (3). Confirm the topic to be processed in Phase 2.
5. Generate new data to be processed based on the problem identified.
6. Study and modify the software processing code on the problem identified to work on the new data.
7. Execute the code in item(6) on the Apache Hadoop, Apache Spark and MLlib infrastructure.
8. Report the results of item(7). Make some changes in data and report the results. Compare the results also with the example provided with the MLlib package.

## 5.3 Dataset Accounting Using Terminal and Java

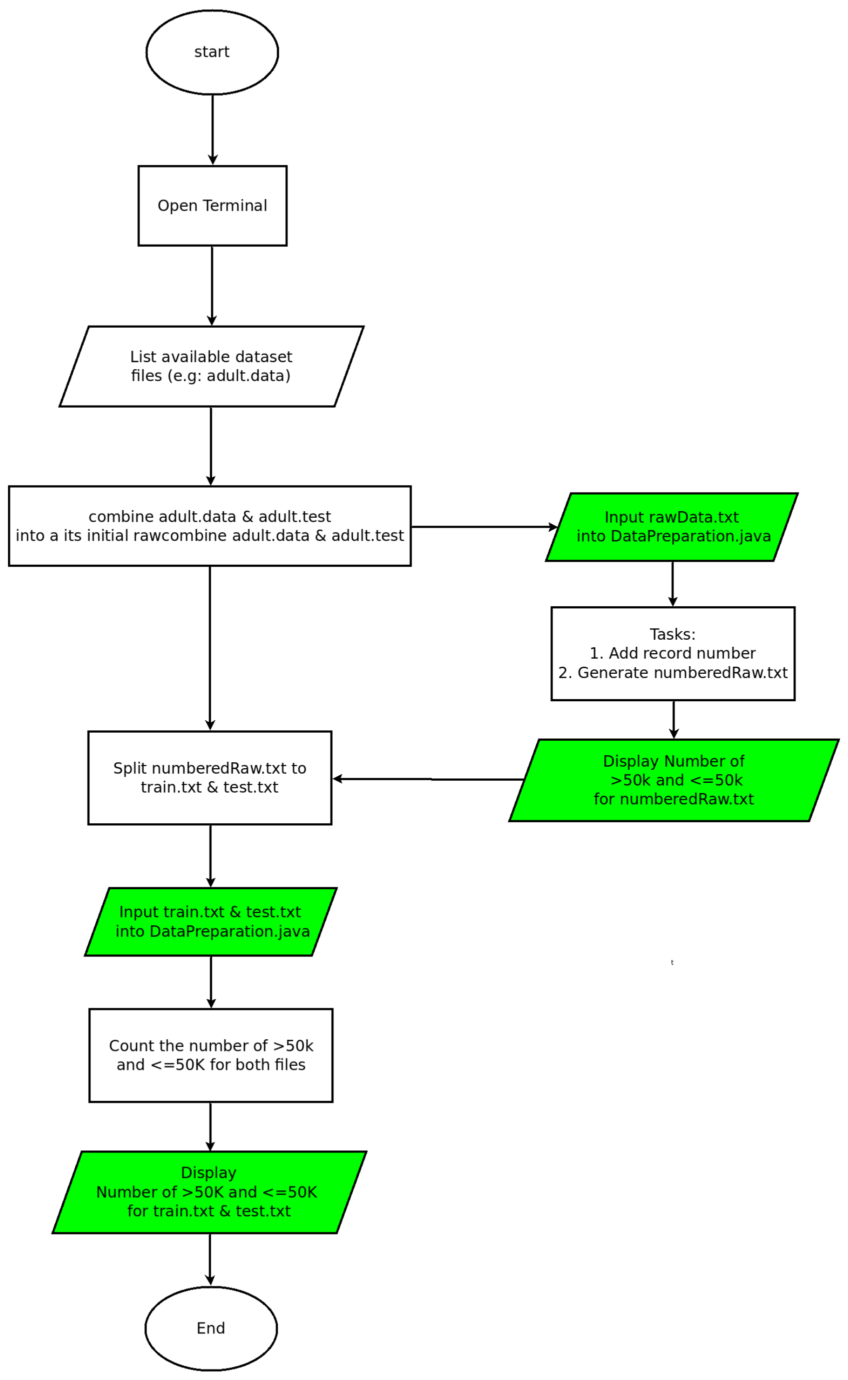

start
Open Terminal
List available dataset files (e.g: adult.data)
combine adult.data & adult.test into a its initial rawcombine adult.data & adult.test
Input rawData.txt into DataPreparation.java
Tasks:
1. Add record number
2. Generate numberedRaw.txt
Display Number of >50k and <=50k for numberedRaw.txt
Split numberedRaw.txt to train.txt & test.txt
Input train.txt & test.txt into DataPreparation.java
Count the number of >50k and <=50K for both files
Display Number of >50K and <=50K for train.txt & test.txt
End

Figure 5.4: Flowchart of Using Terminal and Java for data manipulation

In the figure 5.4 we have put the steps of preparing the dataset into flowchart form. We need to use terminal to produce files and also write a Java program to get files after we create them in Terminal and perform our data preparation tasks. For the code refer to H

## 5.4 Binary Classification Program Flowchart

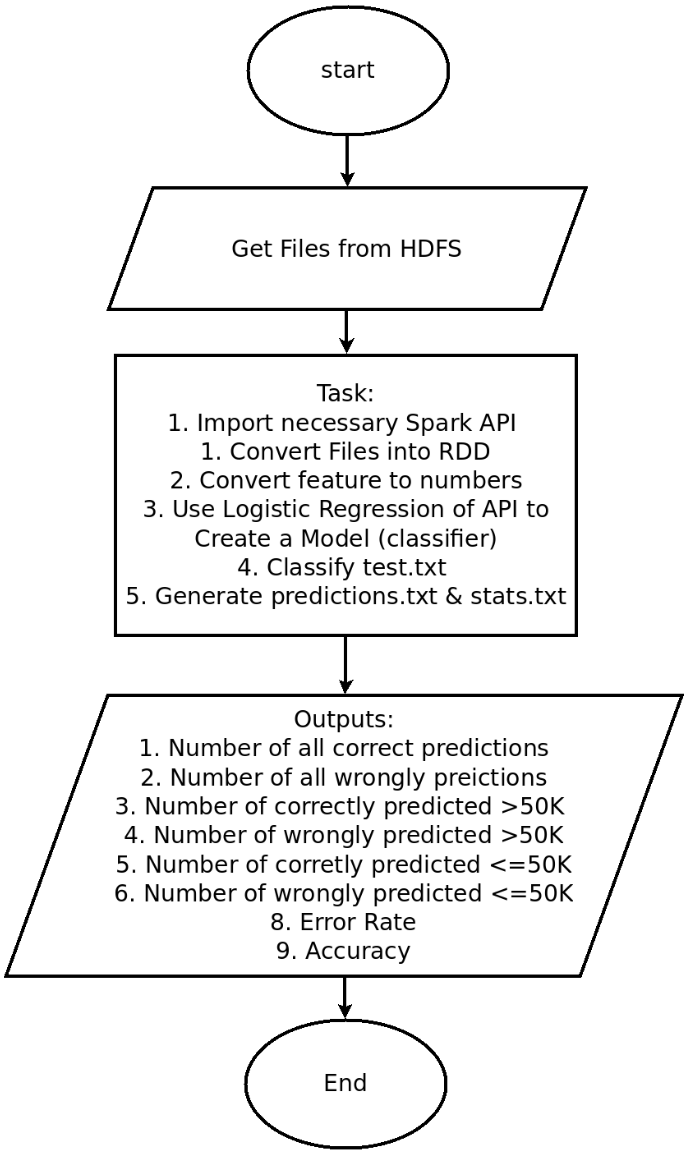

start
Get Files from HDFS
Task:
1. Import necessary Spark API
1. Convert Files into RDD
2. Convert feature to numbers
3. Use Logistic Regression of API to Create a Model (classifier)
4. Classify test.txt
5. Generate predictions.txt & stats.txt
Outputs:
1. Number of all correct predictions
2. Number of all wrongly preictions
3. Number of correctly predicted >50K
4. Number of wrongly predicted >50K
5. Number of corretly predicted <=50K
6. Number of wrongly predicted <=50K
8. Error Rate
9. Accuracy
End

FIGURE 5.5: Flowchart of Binary Classifier Program

Figure 5.5 shows the flow of Java program that used Spark API. The code for this flow chart will be available Appendix H

# Chapter 6

# Testing, Findings, Results, And Comparisons

## 6.1 Single Node Hadoop version 2.4.1

In this section we will be explain how to install Hadoop 2.4.1 on single node machine (Normal Computer). This version of Hadoop was released on June 30,2014.

### 6.1.1 Prerequisites

**Supported OS Platforms**

- GNU/Linux is supported as a development and production platform. Hadoop has been demonstrated on GNU/Linux clusters with 2000 nodes.[12]
- Windows is also a supported platform but the followings steps are for Linux only. To set up Hadoop on Windows, see http://wiki.apache.org/hadoop/Hadoop2OnWin [12]

  For the purpose of this project we are using Ubuntu 14.04, 64-bit. We are not going to explain how to install Ubuntu 14.04 since it is beyond the scope of this project.

**Required Software for Linux**

- Since Hadoop is written in Java the first and most important element will be Java. In order to use Hadoop we need JDK 6+.
- SSH must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.[12]
- Although in many tutorials on Internet you will find that they need a dedicated user account on your local machine for the sake of this FYP we will not going to create a dedicated user for Hadoop, since it is not compulsory for Hadoop to have a dedicated user account.

### 6.1.2 Installing Hadoop

Once you have the required prerequisites we are ready to install Hadoop on our machine.

Steps on installing Hadoop for the script of this part please refer to appendix A:

1. Download the binary file of Hadoop 2.4.1 from the following URL
   `http://psg.mtu.edu/pub/apache/hadoop/common/hadoop-2.4.1/`
   most probably the binary file is in compressed form (e.g: hadoop-2.4.1.tar.gz)
2. Once the download is complete uncompress the file and copy it into the following directory
   /usr/local/
3. At this step your Hadoop is installed on your local machine.[1]

### 6.1.3 Environment Settings

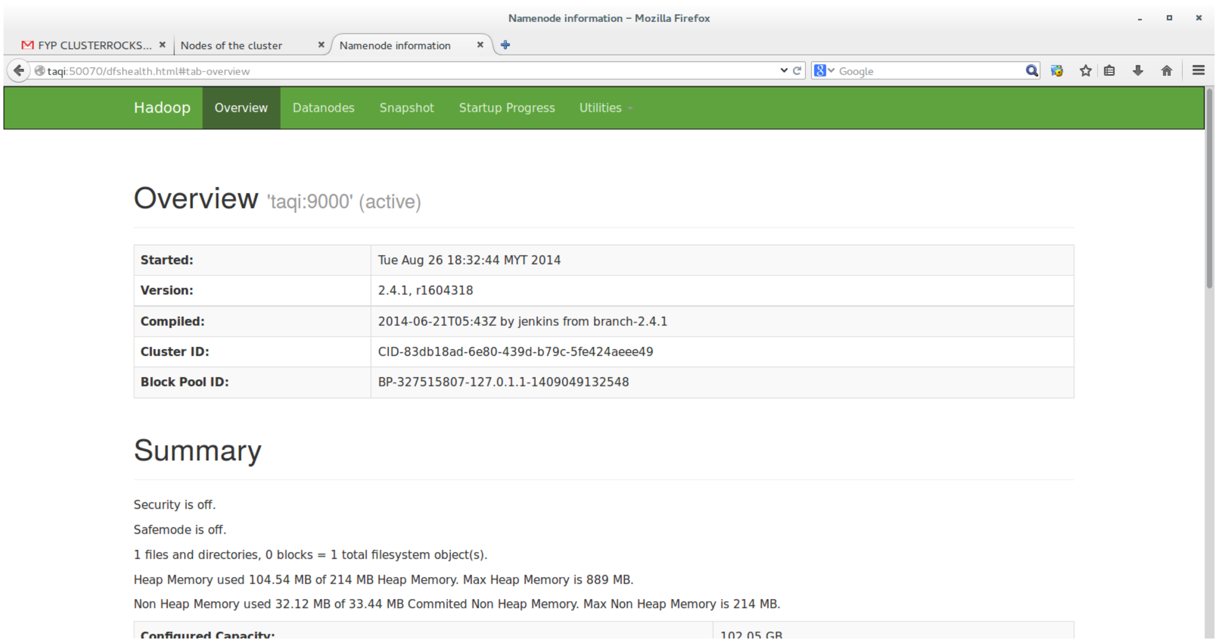
In order to avoid tedious task of going to the home directory of installation each time we want to run the Hadoop is it better to set some environmental variables in our bash.bashrc file to run the Hadoop without needing to go directories of installation.[2]

### 6.1.4 Running Services

Once you have set all the required variables you need to connect to your local host using secure shell and run Hadoop. Once the Hadoop is running you can control and see status of your HDFS and MapReduce applications by going to the following URLs that leads you to Web UI:

[1] See the scripts on Appendix A to see if you have correctly installed Hadoop on your system.
[2] All necessary environmental settings will be available in appendix A.

The first URL is `http://hostname:50070` for my case my hostname is taqi so need to go to `http://taqi:50070`



Overview 'taqi:9000' (active)

| | |
|---|---|
| **Started:** | Tue Aug 26 18:32:44 MYT 2014 |
| **Version:** | 2.4.1, r1604318 |
| **Compiled:** | 2014-06-21T05:43Z by jenkins from branch-2.4.1 |
| **Cluster ID:** | CID-83db18ad-6e80-439d-b79c-5fe424aeee49 |
| **Block Pool ID:** | BP-327515807-127.0.1.1-1409049132548 |

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

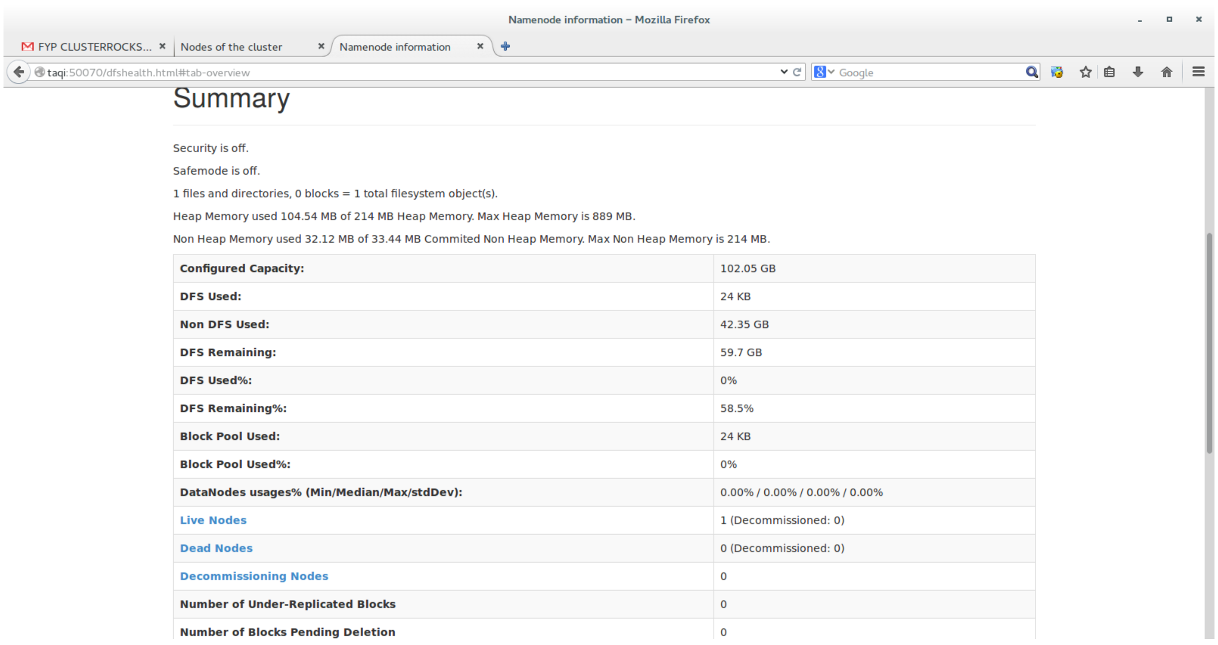
Heap Memory used 104.54 MB of 214 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 32.12 MB of 33.44 MB Commited Non Heap Memory. Max Non Heap Memory is 214 MB.

| | |
|---|---|
| **Configured Capacity:** | 102.05 GB |

FIGURE 6.1: NameNode On Single Node Machine

In the figure 6.1 we can see an overview of our NameNode and some other informations such as when did Hadoop start working and what version of Hadoop we are using and some more information about our NameNode.


Namenode information – Mozilla Firefox
FYP CLUSTERROCKS...
Nodes of the cluster
Namenode information
taqi:50070/dfshealth.html#tab-overview
Google

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

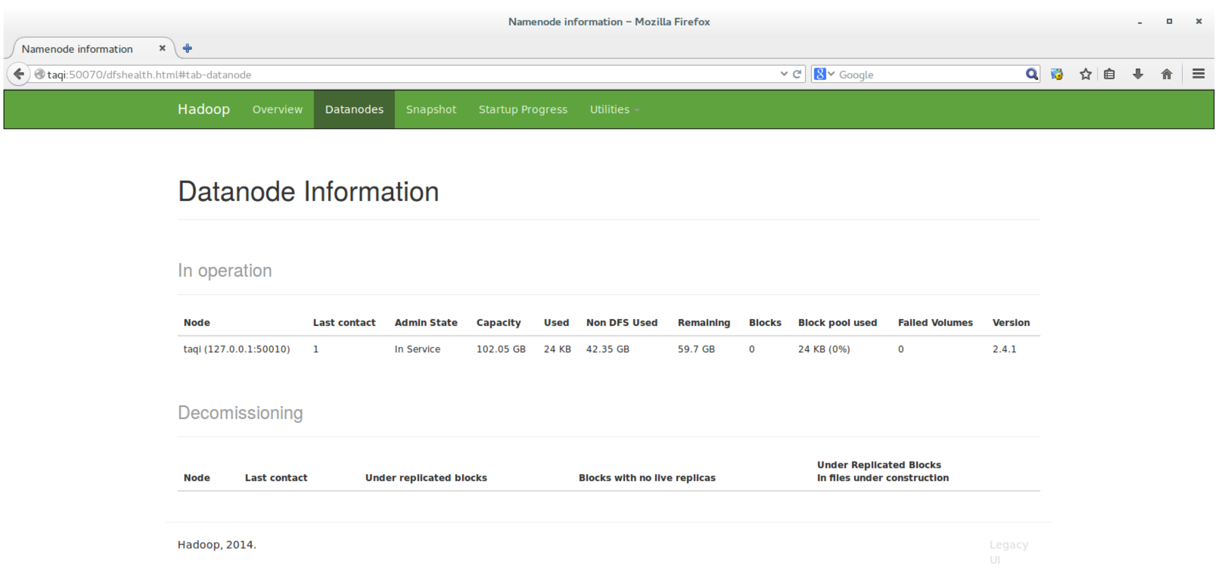
Heap Memory used 104.54 MB of 214 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 32.12 MB of 33.44 MB Commited Non Heap Memory. Max Non Heap Memory is 214 MB.

| | |
|---|---|
| **Configured Capacity:** | 102.05 GB |
| **DFS Used:** | 24 KB |
| **Non DFS Used:** | 42.35 GB |
| **DFS Remaining:** | 59.7 GB |
| **DFS Used%:** | 0% |
| **DFS Remaining%:** | 58.5% |
| **Block Pool Used:** | 24 KB |
| **Block Pool Used%:** | 0% |
| **DataNodes usages% (Min/Median/Max/stdDev):** | 0.00% / 0.00% / 0.00% / 0.00% |
| **Live Nodes** | 1 (Decommissioned: 0) |
| **Dead Nodes** | 0 (Decommissioned: 0) |
| **Decommissioning Nodes** | 0 |
| **Number of Under-Replicated Blocks** | 0 |
| **Number of Blocks Pending Deletion** | 0 |

FIGURE 6.2: Summary of NameNode On Single Node Machine

Figure 6.2 shows a summary of our entire HDFS the number of live, dead nodes, and etc. Since we are running on single node machine we only got one live node and 0 dead nodes which is good sign and it means we have successfully configured Hadoop in our single node machine.



Datanode Information

In operation

| Node | Last contact | Admin State | Capacity | Used | Non DFS Used | Remaining | Blocks | Block pool used | Failed Volumes | Version |
|---|---|---|---|---|---|---|---|---|---|---|
| taqi (127.0.0.1:50010) | 1 | In Service | 102.05 GB | 24 KB | 42.35 GB | 59.7 GB | 0 | 24 KB (0%) | 0 | 2.4.1 |

Decomissioning

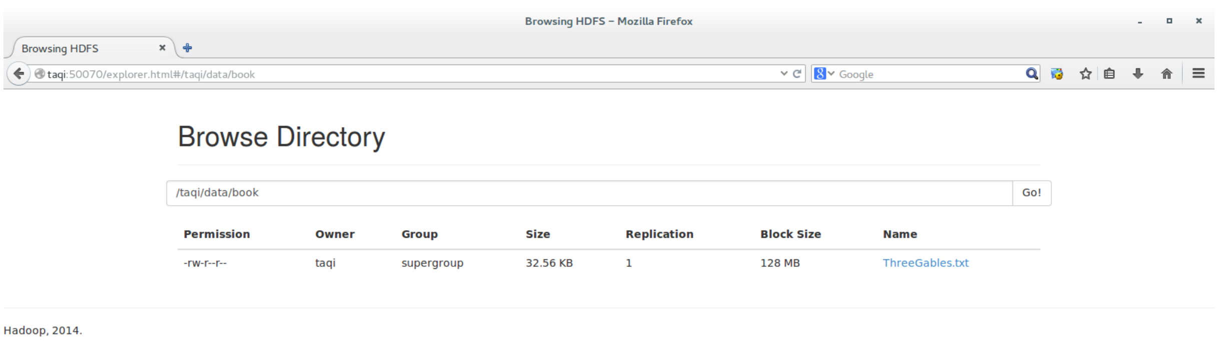
| Node | Last contact | Under replicated blocks | Blocks with no live replicas | Under Replicated Blocks In files under construction |
|---|---|---|---|---|

Hadoop, 2014.

Legacy UI

FIGURE 6.3: DataNodes Information On Single Node Machine

In this figure we can see information about our Datanode or Datanodes as we go to multi-nodes cluster.


Browsing HDFS – Mozilla Firefox
Browsing HDFS
taqi:50070/explorer.html#/taqi/data/book
Google

Browse Directory

/taqi/data/book Go!

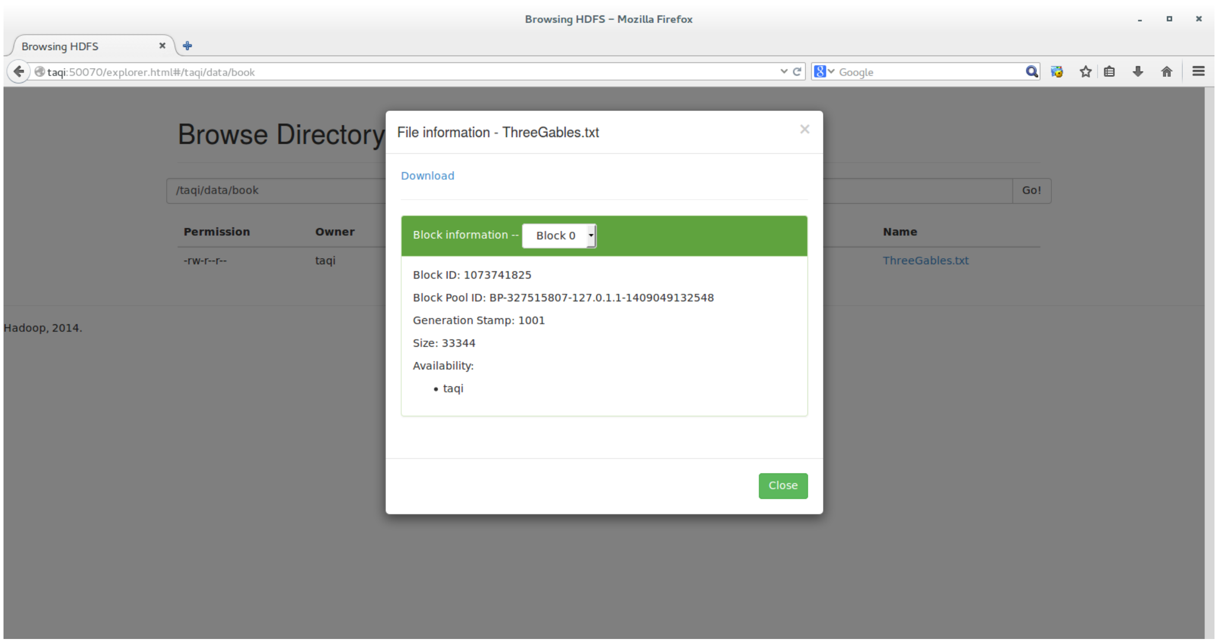
| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | taqi | supergroup | 32.56 KB | 1 | 128 MB | ThreeGables.txt |

Hadoop, 2014.

FIGURE 6.4: Files Stored in HDFS On Single Node Machine

After copying files into your directory you can see them as one files as you see in normal file system but the fact it has been split out into storage blocks of. You will see more clear on multi-nodes cluster.It is the Web UI that shows this as one file for purpose of user friendliness.

Once we have our file stored in HDFS we are ready to run our MapReduce application against it. See Apendix A for "Running WordCount MapReduce Application" to see how to run a MapReduce application against your HDFS. Once the WordCount has done its work it will save the result into the new directory as Figure 6.6.


Browsing HDFS – Mozilla Firefox

taqi:50070/explorer.html#/taqi/data/book

Browse Directory

/taqi/data/book

| Permission | Owner | Name |
|---|---|---|
| -rw-r--r-- | taqi | ThreeGables.txt |

Hadoop, 2014.

File information - ThreeGables.txt

Download

Block information -- Block 0

Block ID: 1073741825

Block Pool ID: BP-327515807-127.0.1.1-1409049132548

Generation Stamp: 1001

Size: 33344

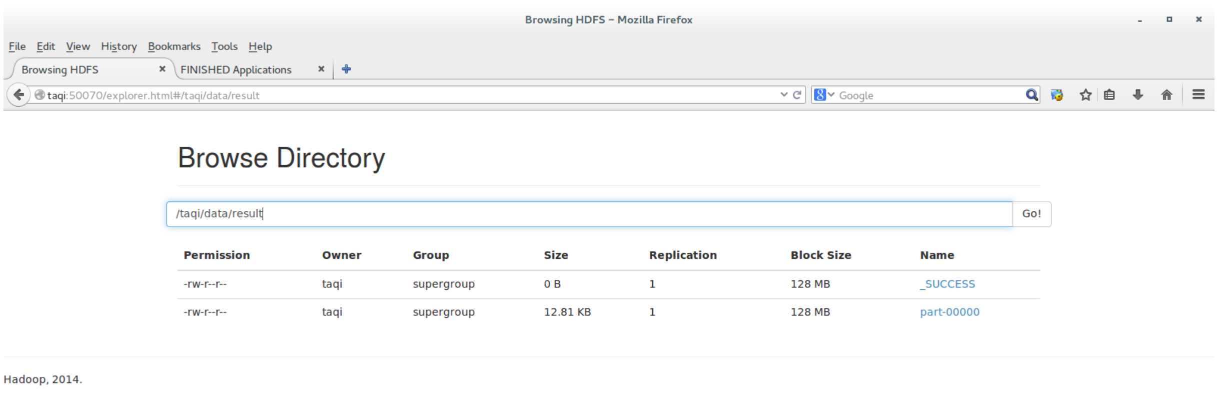
Availability:

- taqi

Close

FIGURE 6.5: Download Files Stored in HDFS On Single Node Machine

The above figure shows enable user to download their files back from HDFS.


Browsing HDFS – Mozilla Firefox

File Edit View History Bookmarks Tools Help

Browsing HDFS | FINISHED Applications

taqi:50070/explorer.html#/taqi/data/result

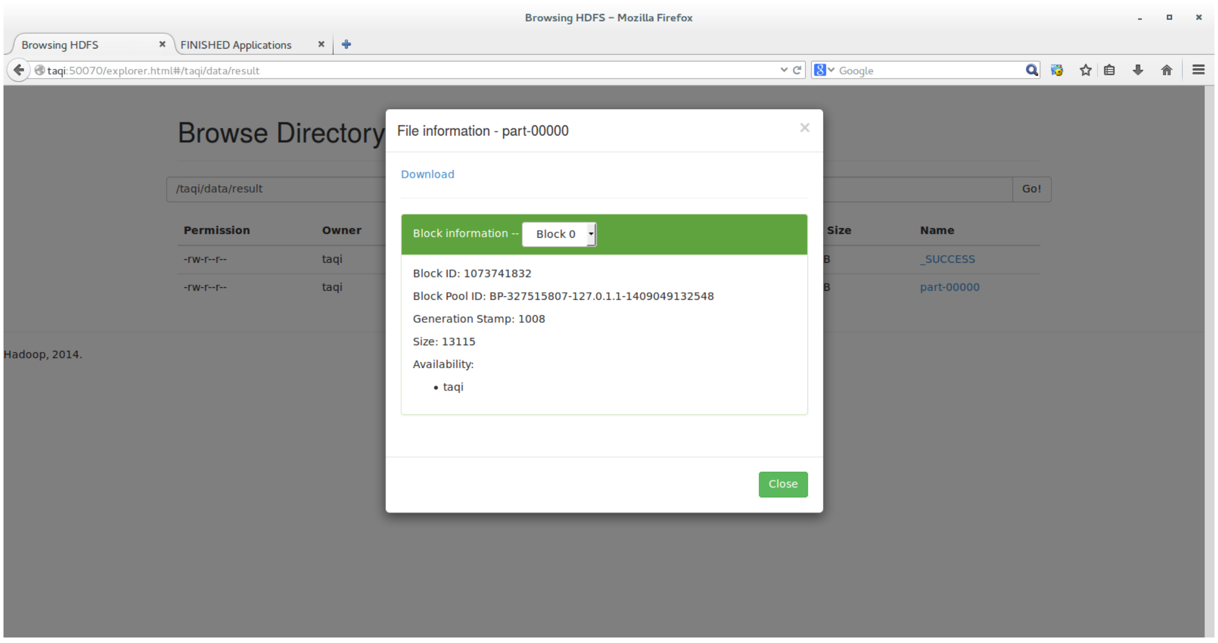
Browse Directory

/taqi/data/result Go!

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | taqi | supergroup | 0 B | 1 | 128 MB | _SUCCESS |
| -rw-r--r-- | taqi | supergroup | 12.81 KB | 1 | 128 MB | part-00000 |

Hadoop, 2014.

FIGURE 6.6: Result Files Stored in HDFS On Single Node Machine


Browsing HDFS – Mozilla Firefox

File information - part-00000

Download

Block information -- Block 0

Block ID: 1073741832

Block Pool ID: BP-327515807-127.0.1.1-1409049132548

Generation Stamp: 1008

Size: 13115

Availability:

- taqi

Close

FIGURE 6.7: Download Result Files Stored in HDFS On Single Node Machine

### 6.1.5 Example of Using Hadoop

After running all 5 services of Hadoop we will run a word count in MapReduce model as prototype to get familiar with the method of programming for Hadoop and MapReduce model.

Every program written in MapReduce model should be consists of two main method, Map and Reduce. The source code of our WordCount application has been taken from Apache website and been provided on Appendix B.

Word count basically is an application that counts the number of occurrences of a word in given text file. But its difference with a normal word count is that it is based on MapReduce programming model. This means that our word count problem is consists of two major parts that each MapReduce application needs to have , Map and Reduce functions.

Our WordCount.java is consist of two subclass and one main method:

- Map: it inherits MapReduceBase class of MapReduce library and acts as an interface to implement the Mapper part of that class. Our input in this class is a text file and we take each line as $< key, value >$ pair where $< ByteOffSet, EntireLine >$ and output $< word, 1 >$ . This class will be sent to each Datanode to process the data.
- Reduce: it inherits MapReduceBase class of MapReduce library and acts as an interface to implement the Reduce part of that class. Our input in this class is $< key, value >$ pairs of Map function ,$< word, 1 >$, and output is $< Word, SumOfOccurence >$ This class will get the output of combiner in iteration mode and produce the final result.
- Main: We setup a MapReduce job and name it ”wordcounter”. We will set all the necessary elements of a job such as the type of input, output, key, value and the format of input and output files and also the path of our input as well as the directory that we want our output be stored.

### 6.1.6 Proof of Concept

For the proof of concept we intentionally put our names with in text file to examine and force the program to display our desired output. After putting our name in the input file we get the exact result of their occurrence in the result file and the sample output has been provided in appendix ??.

### 6.1.7 Summary

Summary of this section:

1. How to install Hadoop on single node cluster and run it
2. How to store files from local storage into Hadoop Distributed File System (HDFS).
3. How to write a MapReduce application
4. How to run a MapReduce application on HDFS.

## 6.2 Multi-Node Cluster Hadoop version 2.4.1

In this section we will expand our understanding of Hadoop installation on single node machine to a wider area and will see how to setup and configure Hadoop-2.4.1 on cluster of computers.

The reason we will be installing Hadoop-2.4.1 is that we already know how to install it on single machine and this understanding will help us in installing it on cluster. We will be doing this installation on clusterrocks that locates on FCI Grid lab.

### 6.2.1 Prerequisites

Prerequisites for installing Hadoop on cluster of inexpensive commodity machines are as the same as installing it on single one. We need to have Java in all of our machines and Master and Slaves should have same ssh key.

### 6.2.2 Environment Settings

Environment settings like the single node installation is optional and it is for the convenient of user and it is not mandatory to do. However if you still wish to set the environment settings refer to Single Node Installation. In addition to this we need rsync package for syncing all of our slaves.

### 6.2.3 Installing Hadoop

Installing Hadoop on cluster of machines is basically copying the same installation file into all of the machines.

In order to install Hadoop on Multi-node cluster we need to do the following, for the scripts please refer to appendix B:

1. Download the binary file of Hadoop 2.4.1 from the following URL
   `http://psg.mtu.edu/pub/apache/hadoop/common/hadoop-2.4.1/`
   most probably the binary file is in compressed form (e.g: hadoop-2.4.1.tar.gz)
2. Once the download is complete uncompress the file and copy it into the following directory of your master node
   `/usr/local/`
3. We need to create two files ,Master and Slaves, in configuration folder to tell Hadoop which node is Master and which nodes should be used as Slaves. It is quite easy because we only need to type the name of master node in file

called masters and type the name of all salves in separated lines in another file called slaves and place it in this directory `/usr/local/hadoop-2.4.1` of you master node.

4. Setup the configuration files in `/usr/local/hadoop-2.4.1/etc/hadoop`
5. At this step your Hadoop is installed and ready to run on your Master node you only need to copy the installation files into your slave nodes and run you Hadoop.[3]

### 6.2.4 Running Services

Since Hadoop require super (root) privilege for some of its services on slave nodes we decided to run all steps as root user.

Steps to run Hadoop on clusterrocks:

1. Connect to the cluster with as root
2. Go to Hadoop directory
3. Format Namenode if it is the first time running Hadoop on cluster
4. Run all required services as root
5. Make sure you have all needed services up and running

4

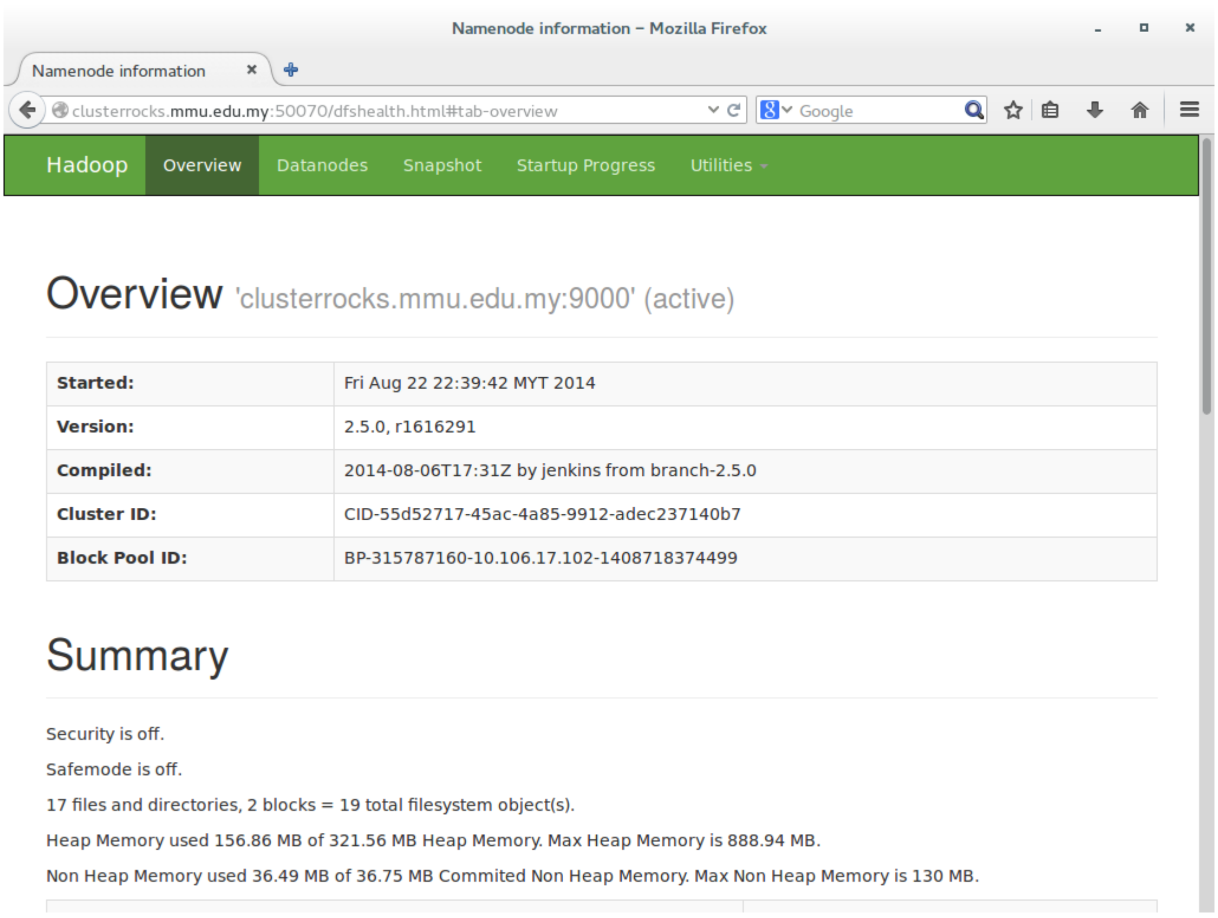
If you have correctly installed and run Hadoop on your cluster you need should be able to see two running service on all of your salve nodes , NodeManager

[3]See the scripts on Appendix A on to see if you have correctly installed Hadoop on your system

[4]See Appendix A for scripts running Hadoop-2.4.1 on clusterrocks

and DataNode, and five other service running on you master node which are DataNode, JobHistoryServer, NodeManager, ResourceManager .

Now if you go to the Web UI in the below links you should be able to see an interface of Hadoop on your browser. `clusterrocks.mmu.edu.my:50070`


Namenode information – Mozilla Firefox
Namenode information
clusterrocks.mmu.edu.my:50070/dfshealth.html#tab-overview
Google
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities

Overview 'clusterrocks.mmu.edu.my:9000' (active)

| | |
|---|---|
| **Started:** | Fri Aug 22 22:39:42 MYT 2014 |
| **Version:** | 2.5.0, r1616291 |
| **Compiled:** | 2014-08-06T17:31Z by jenkins from branch-2.5.0 |
| **Cluster ID:** | CID-55d52717-45ac-4a85-9912-adec237140b7 |
| **Block Pool ID:** | BP-315787160-10.106.17.102-1408718374499 |

Summary

Security is off.

Safemode is off.

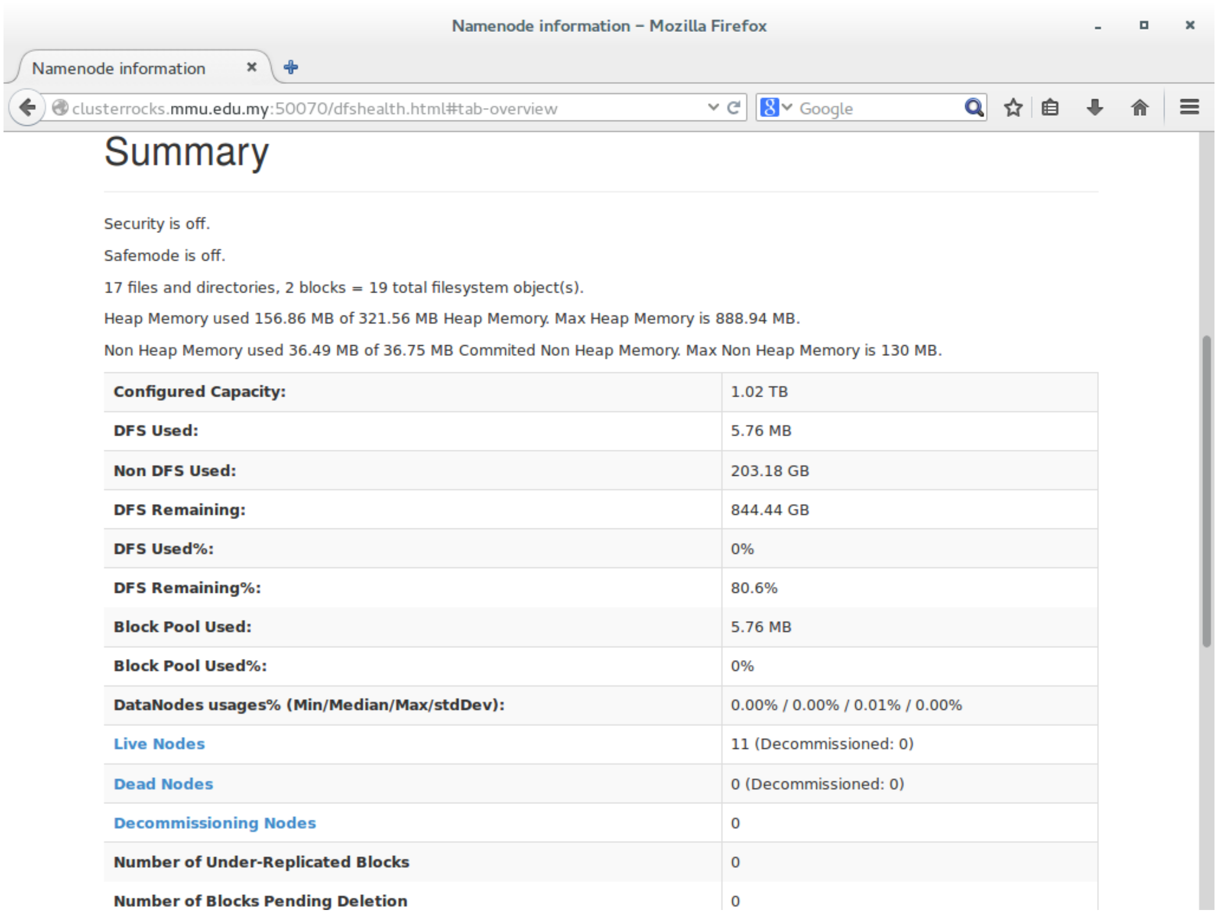
17 files and directories, 2 blocks = 19 total filesystem object(s).

Heap Memory used 156.86 MB of 321.56 MB Heap Memory. Max Heap Memory is 888.94 MB.

Non Heap Memory used 36.49 MB of 36.75 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

FIGURE 6.8: NameNode On Clusterrocks

In the figure above we can see an overview of our NameNode and some other informations such as when did Hadoop start working and what version of Hadoop we are using and some more information about our NameNode.


Namenode information – Mozilla Firefox
Namenode information
clusterrocks.mmu.edu.my:50070/dfshealth.html#tab-overview
Google

Summary

Security is off.

Safemode is off.

17 files and directories, 2 blocks = 19 total filesystem object(s).

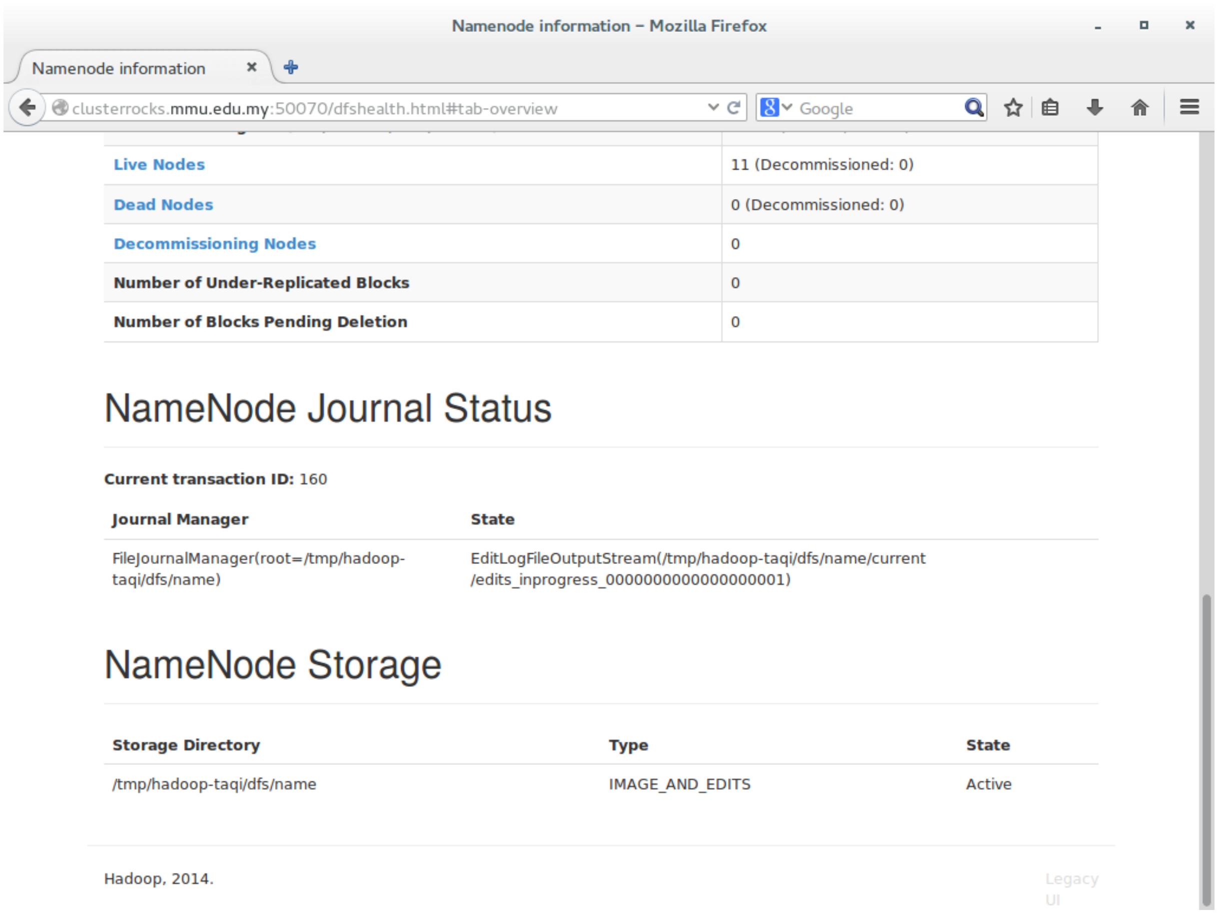
Heap Memory used 156.86 MB of 321.56 MB Heap Memory. Max Heap Memory is 888.94 MB.

Non Heap Memory used 36.49 MB of 36.75 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

| | |
|---|---|
| **Configured Capacity:** | 1.02 TB |
| **DFS Used:** | 5.76 MB |
| **Non DFS Used:** | 203.18 GB |
| **DFS Remaining:** | 844.44 GB |
| **DFS Used%:** | 0% |
| **DFS Remaining%:** | 80.6% |
| **Block Pool Used:** | 5.76 MB |
| **Block Pool Used%:** | 0% |
| **DataNodes usages% (Min/Median/Max/stdDev):** | 0.00% / 0.00% / 0.01% / 0.00% |
| **Live Nodes** | 11 (Decommissioned: 0) |
| **Dead Nodes** | 0 (Decommissioned: 0) |
| **Decommissioning Nodes** | 0 |
| **Number of Under-Replicated Blocks** | 0 |
| **Number of Blocks Pending Deletion** | 0 |

Figure 6.9: Summary of NameNode On Clusterrocks

This figure shows a summary of our entire HDFS the number of live, dead nodes, and etc. Since we are running on cluster of machine we can see the total capacity has increased and now we have 11 live nodes and it is a good sign which mean all of our node (1 master and 10 slaves) are in good condition.


Namenode information – Mozilla Firefox
Namenode information
clusterrocks.mmu.edu.my:50070/dfshealth.html#tab-overview
Google

| | |
|---|---|
| Live Nodes | 11 (Decommissioned: 0) |
| Dead Nodes | 0 (Decommissioned: 0) |
| Decommissioning Nodes | 0 |
| Number of Under-Replicated Blocks | 0 |
| Number of Blocks Pending Deletion | 0 |

NameNode Journal Status

**Current transaction ID:** 160

| Journal Manager | State |
|---|---|
| FileJournalManager(root=/tmp/hadoop-taqi/dfs/name) | EditLogFileOutputStream(/tmp/hadoop-taqi/dfs/name/current/edits_inprogress_0000000000000000001) |

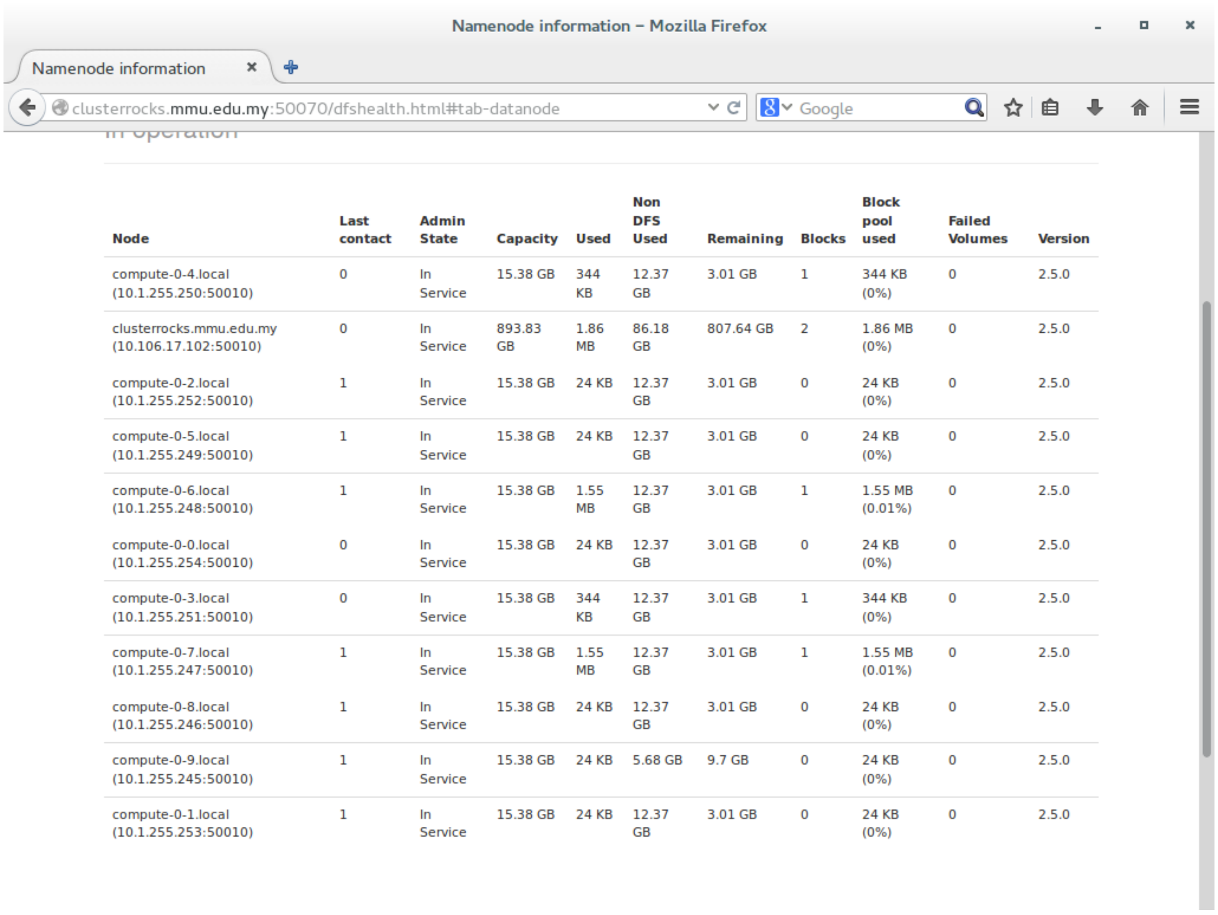
NameNode Storage

| Storage Directory | Type | State |
|---|---|---|
| /tmp/hadoop-taqi/dfs/name | IMAGE_AND_EDITS | Active |

Hadoop, 2014. Legacy UI

FIGURE 6.10: Summary of NameNode on Clusterrocks-2

In the figure above we will see more information about our Hadoop such as what is the directory of Namenode and storage directory of NameNode.

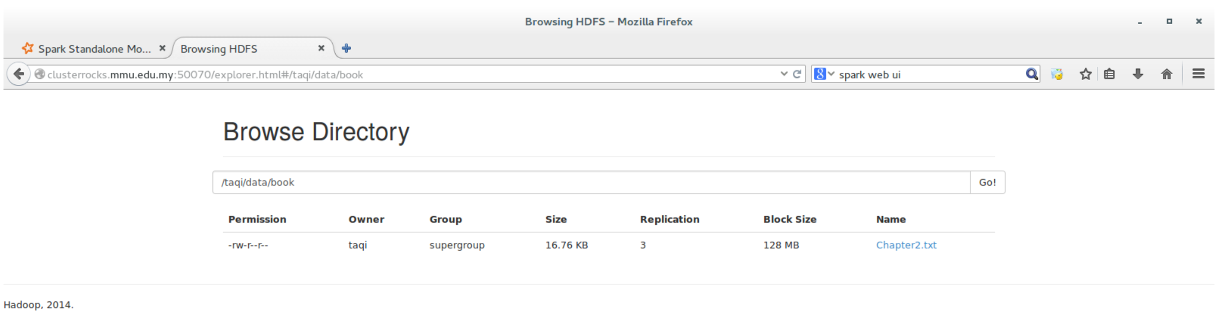

Namenode information – Mozilla Firefox
Namenode information
clusterrocks.mmu.edu.my:50070/dfshealth.html#tab-datanode
Google

| Node | Last contact | Admin State | Capacity | Used | Non DFS Used | Remaining | Blocks | Block pool used | Failed Volumes | Version |
|---|---|---|---|---|---|---|---|---|---|---|
| compute-0-4.local (10.1.255.250:50010) | 0 | In Service | 15.38 GB | 344 KB | 12.37 GB | 3.01 GB | 1 | 344 KB (0%) | 0 | 2.5.0 |
| clusterrocks.mmu.edu.my (10.106.17.102:50010) | 0 | In Service | 893.83 GB | 1.86 MB | 86.18 GB | 807.64 GB | 2 | 1.86 MB (0%) | 0 | 2.5.0 |
| compute-0-2.local (10.1.255.252:50010) | 1 | In Service | 15.38 GB | 24 KB | 12.37 GB | 3.01 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |
| compute-0-5.local (10.1.255.249:50010) | 1 | In Service | 15.38 GB | 24 KB | 12.37 GB | 3.01 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |
| compute-0-6.local (10.1.255.248:50010) | 1 | In Service | 15.38 GB | 1.55 MB | 12.37 GB | 3.01 GB | 1 | 1.55 MB (0.01%) | 0 | 2.5.0 |
| compute-0-0.local (10.1.255.254:50010) | 0 | In Service | 15.38 GB | 24 KB | 12.37 GB | 3.01 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |
| compute-0-3.local (10.1.255.251:50010) | 0 | In Service | 15.38 GB | 344 KB | 12.37 GB | 3.01 GB | 1 | 344 KB (0%) | 0 | 2.5.0 |
| compute-0-7.local (10.1.255.247:50010) | 1 | In Service | 15.38 GB | 1.55 MB | 12.37 GB | 3.01 GB | 1 | 1.55 MB (0.01%) | 0 | 2.5.0 |
| compute-0-8.local (10.1.255.246:50010) | 1 | In Service | 15.38 GB | 24 KB | 12.37 GB | 3.01 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |
| compute-0-9.local (10.1.255.245:50010) | 1 | In Service | 15.38 GB | 24 KB | 5.68 GB | 9.7 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |
| compute-0-1.local (10.1.255.253:50010) | 1 | In Service | 15.38 GB | 24 KB | 12.37 GB | 3.01 GB | 0 | 24 KB (0%) | 0 | 2.5.0 |

Figure 6.11: DataNodes Information On Clusterrocks

In figure 6.11 we can see information about our Datanodes.

We can see it is totally different from single node. All information about every slave has been displayed such as how much is their capacity how much of it been used, how many blocks each salve is holding and etc.


Browsing HDFS – Mozilla Firefox
Spark Standalone Mo...
Browsing HDFS
clusterrocks.mmu.edu.my:50070/explorer.html#/taqi/data/book
spark web ui

Browse Directory

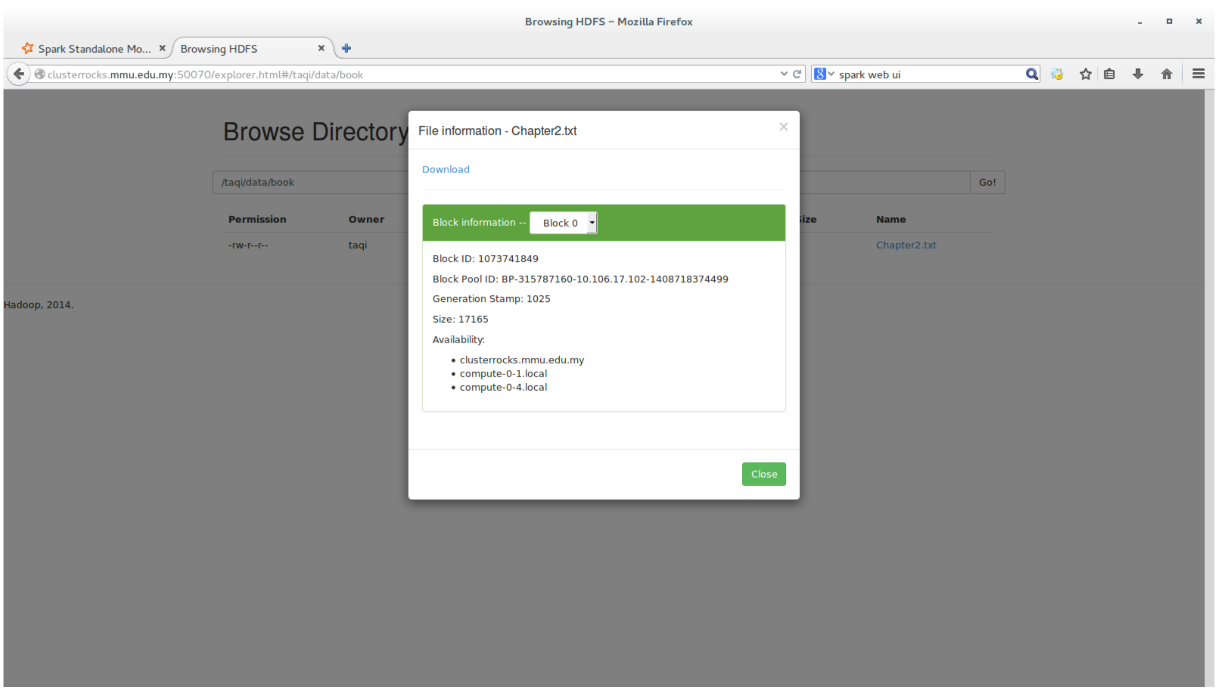
/taqi/data/book Go!

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | taqi | supergroup | 16.76 KB | 3 | 128 MB | Chapter2.txt |

Hadoop, 2014.

FIGURE 6.12: Files Stored in HDFS On Clusterrocks

After copying files into your directory you can see them as one files as you see in normal file system but the fact it has been split out into storage blocks of. You will see more clear on multi-nodes cluster.It is the Web UI that shows this as one file for purpose of user friendliness.
Once we have our file stored in HDFS we are ready to run our MapReduce application against it. See Apendix A for "Running WordCount MapReduce Application" to see how to run a MapReduce application against your HDFS. Once the WordCount has done its work it will save the result into the new directory as Figure below.



File information - Chapter2.txt

Download

Block information -- Block 0

| Permission | Owner | Size | Name |
|---|---|---|---|
| -rw-r--r-- | taqi | | Chapter2.txt |

Block ID: 1073741849

Block Pool ID: BP-315787160-10.106.17.102-1408718374499

Generation Stamp: 1025

Size: 17165

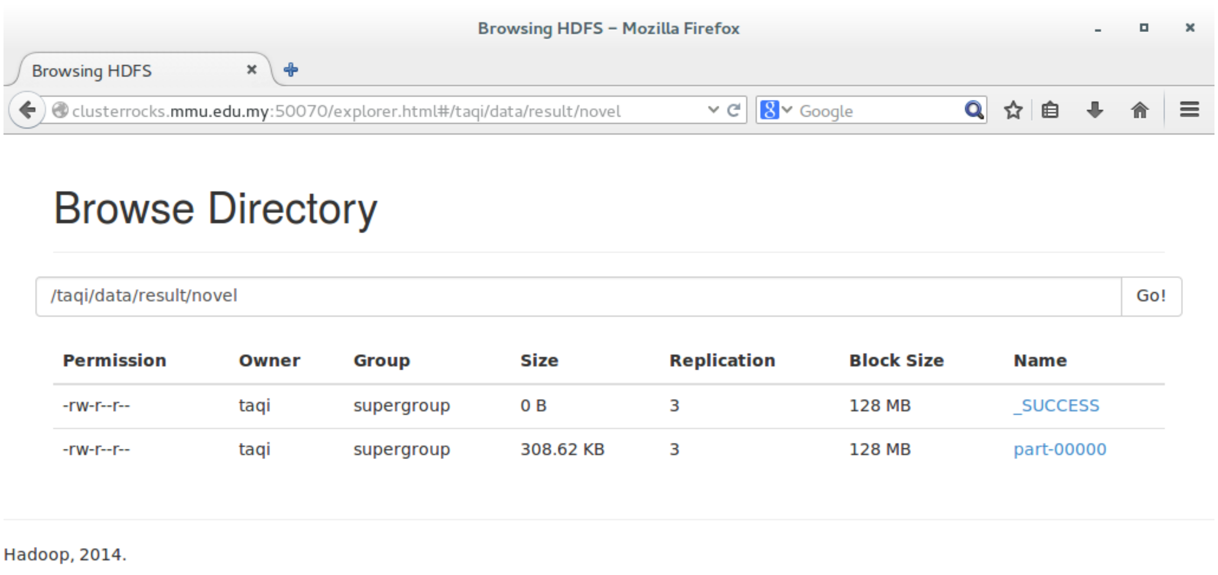
Availability:

- clusterrocks.mmu.edu.my
- compute-0-1.local
- compute-0-4.local

Close

FIGURE 6.13: Download Files Stored in HDFS On Clusterrocks

The figure 6.13 shows enable user to download their files back from HDFS. It will also provide other information such as the nodes that has saved this file's block.


Browsing HDFS – Mozilla Firefox
Browsing HDFS
clusterrocks.mmu.edu.my:50070/explorer.html#/taqi/data/result/novel
Google
Go!

Browse Directory

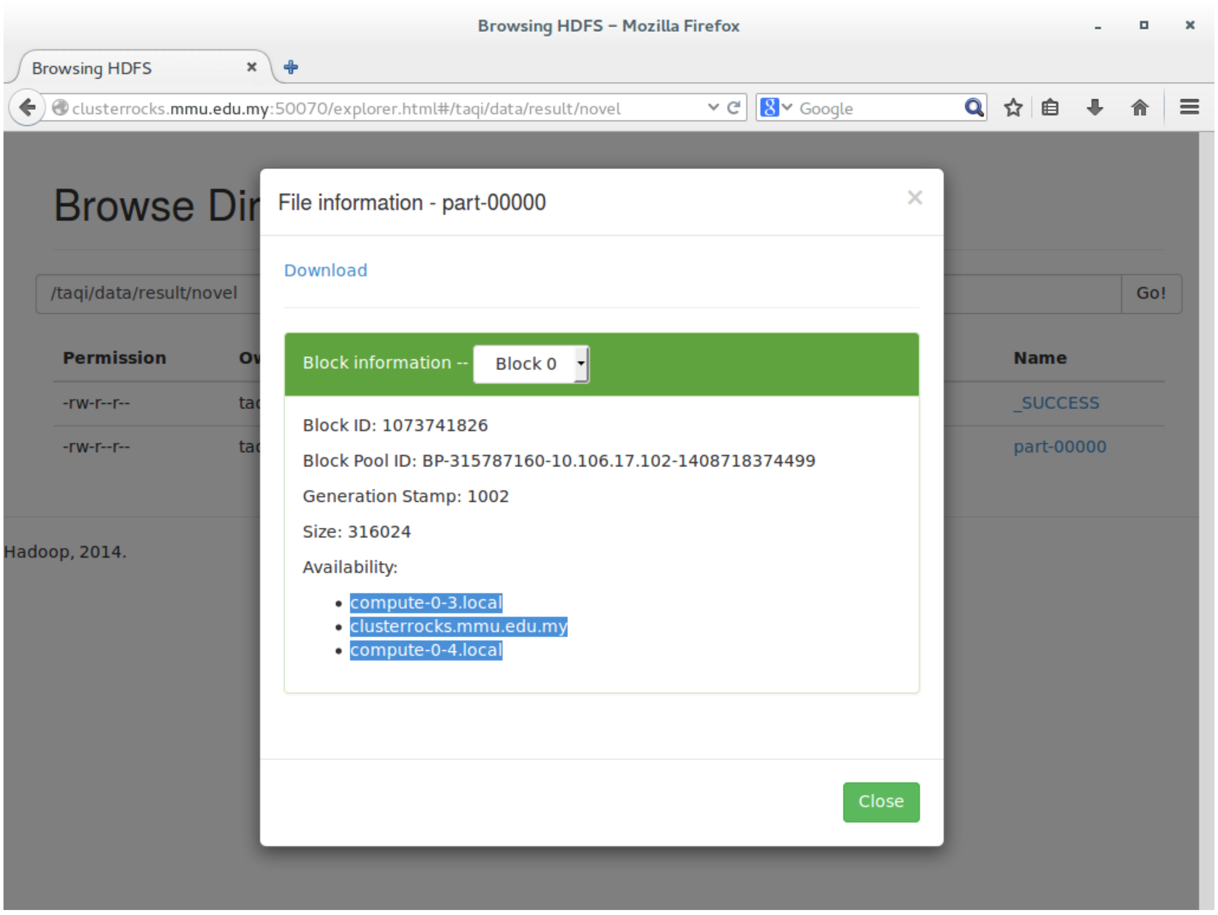
/taqi/data/result/novel

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | taqi | supergroup | 0 B | 3 | 128 MB | _SUCCESS |
| -rw-r--r-- | taqi | supergroup | 308.62 KB | 3 | 128 MB | part-00000 |

Hadoop, 2014.

FIGURE 6.14: Result Files Stored in HDFS On Clusterrocks

After running MapReduce application the result will be stored in different Datanode but the web UI enable us to see them as one file and download it.



File information - part-00000

Block information --

Block ID: 1073741826

Block Pool ID: BP-315787160-10.106.17.102-1408718374499

Generation Stamp: 1002

Size: 316024

Availability:

- compute-0-3.local
- clusterrocks.mmu.edu.my
- compute-0-4.local

FIGURE 6.15: Download Result Files Stored in HDFS On Clusterrocks

### 6.2.5 Example of Using Hadoop

Same as single node machine that we run the word count on Hadoop-2.4.1, we can run the same example on our cluster.

We are not going to discuss the detail of the program since it is exactly same but for the sake of understanding it better we will provide you some screen shots of the web UI to see what actually happens when we run a MapReduce Application on clusterrocks.

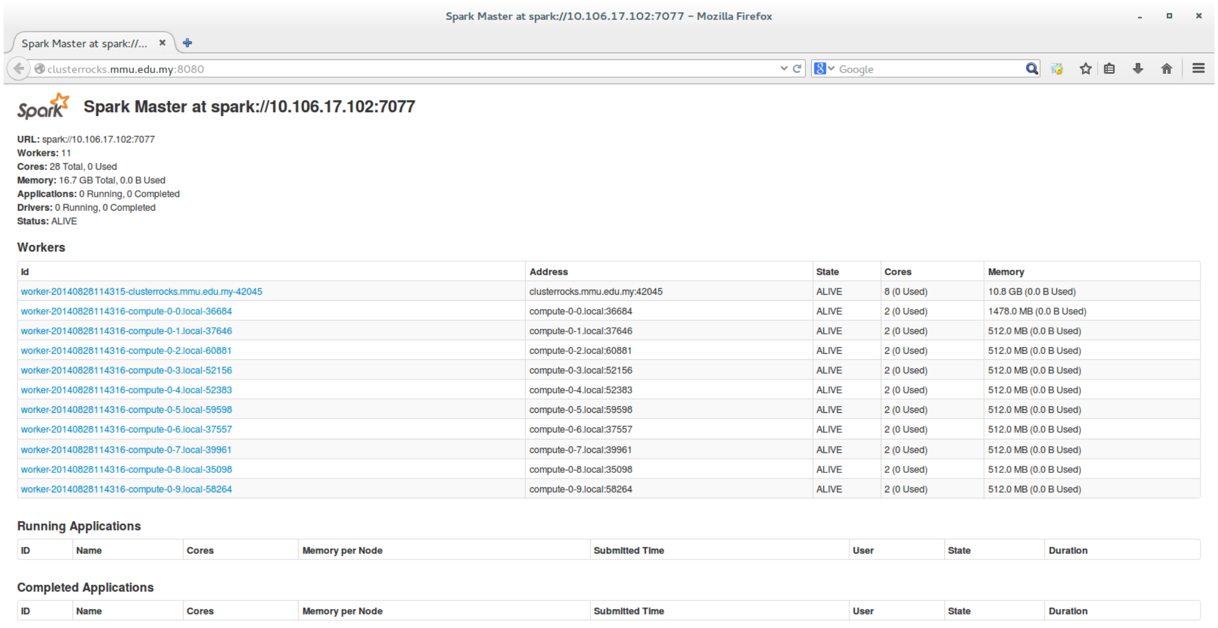
### 6.2.6 Summary

Summary of this section is as below

1. Installing Hadoop-2.4.1 on cluster of machine on FCI Grid lab
2. Run Hadoop on clusterrocks
3. Understand the architecture of Hadoop with the help of Web UI
4. Store and Retrieve files on clusterrocks
5. Run WordCount.java against cluster of machines `clusterrocks.mmu.edu.my`

## 6.3 Installing Spark On Clusterrocks

In figure 6.16 you can see spark running on clusterrocks successfully.


Spark Master at spark://10.106.17.102:7077 – Mozilla Firefox
Spark Master at spark://...
clusterrocks.mmu.edu.my:8080
Google

**Spark Master at spark://10.106.17.102:7077**

**URL:** spark://10.106.17.102:7077
**Workers:** 11
**Cores:** 28 Total, 0 Used
**Memory:** 16.7 GB Total, 0.0 B Used
**Applications:** 0 Running, 0 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

**Workers**

| Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20140828114315-clusterrocks.mmu.edu.my-42045 | clusterrocks.mmu.edu.my:42045 | ALIVE | 8 (0 Used) | 10.8 GB (0.0 B Used) |
| worker-20140828114316-compute-0-0.local-36684 | compute-0-0.local:36684 | ALIVE | 2 (0 Used) | 1478.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-1.local-37646 | compute-0-1.local:37646 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-2.local-60881 | compute-0-2.local:60881 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-3.local-52156 | compute-0-3.local:52156 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-4.local-52383 | compute-0-4.local:52383 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-5.local-59598 | compute-0-5.local:59598 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-6.local-37557 | compute-0-6.local:37557 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-7.local-39961 | compute-0-7.local:39961 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-8.local-35098 | compute-0-8.local:35098 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |
| worker-20140828114316-compute-0-9.local-58264 | compute-0-9.local:58264 | ALIVE | 2 (0 Used) | 512.0 MB (0.0 B Used) |

**Running Applications**

| ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|

**Completed Applications**

| ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|

FIGURE 6.16: Spark running on clusterrocks as root

As you can see in the figure 6.16 we successfully run Spark on clusterrocks computers on FCI Grid lab. We have put the scripts on how to install Spark on Appendix E.
After installing and configuration of Hadoop on single node and Multinode cluster we know are more confident on what we do and have better understanding what each step of installation actually mean.

- Download the installation package from spark website
- Extract files into directory `/usr/local`
- Setup environment settings
- Copy installation package in all other nodes.

## 6.4 Running The Classification program

### 6.4.1 System Requirements

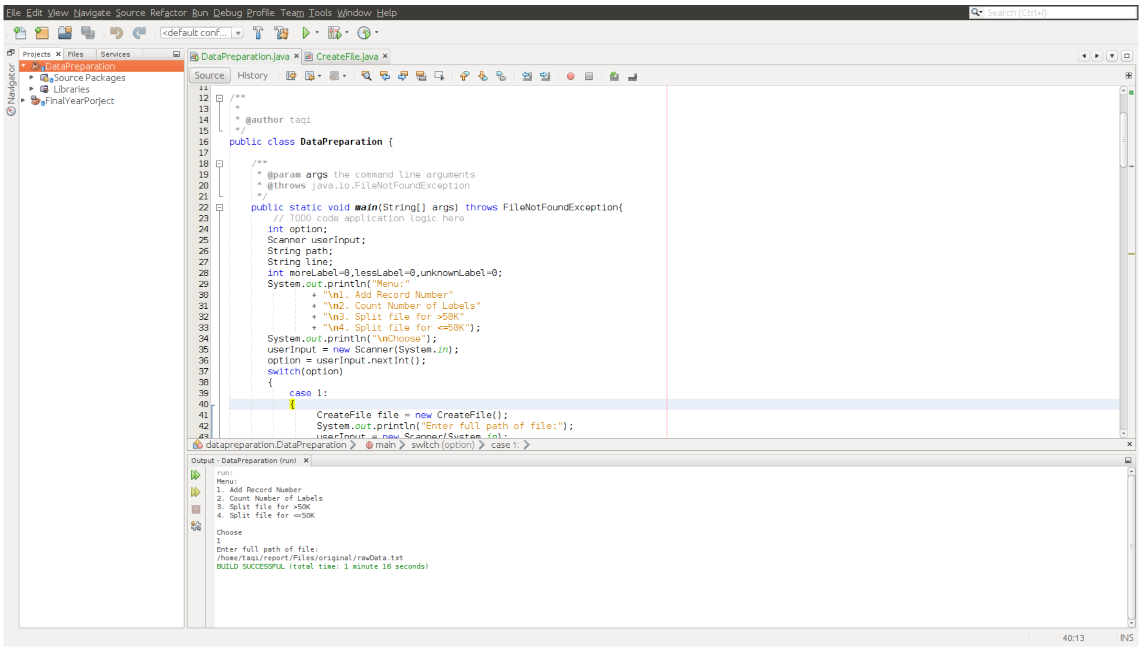
MLlib requires some linear algebra libraries to be installed on machines. First, to use MLlib in any language, we need the Fortran runtime library being installed on our operating system.

We installed Fortran runtime library in our Ubuntu machine with the following command.

```
taqi@taqi-desktop:~$ sudo apt-get install libgfortran3
```

### 6.4.2 Data Preparation

Figure 6.17 is the marking of raw dataset.

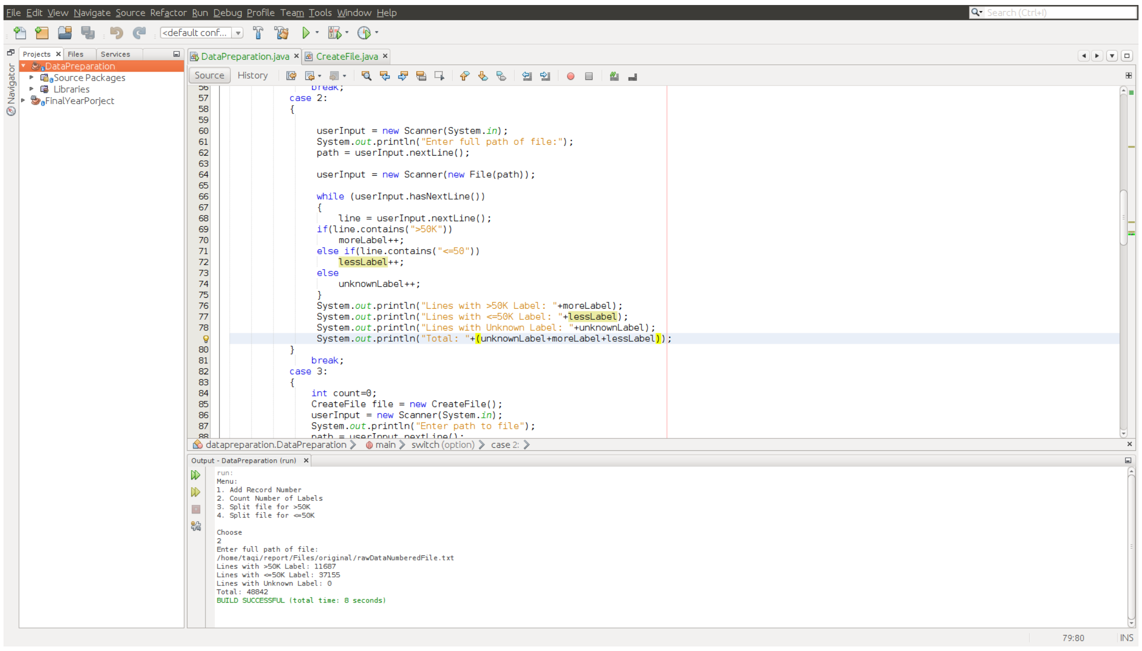


```java
/**
 *
 * @author taqi
 */
public class DataPreparation {

    /**
     * @param args the command line arguments
     * @throws java.io.FileNotFoundException
     */
    public static void main(String[] args) throws FileNotFoundException{
        // TODO code application logic here
        int option;
        Scanner userInput;
        String path;
        String line;
        int moreLabel=0,lessLabel=0,unknownLabel=0;
        System.out.println("Menu:"
                + "\n1. Add Record Number"
                + "\n2. Count Number of Labels"
                + "\n3. Split file for >50K"
                + "\n4. Split file for <=50K");
        System.out.println("\nChoose");
        userInput = new Scanner(System.in);
        option = userInput.nextInt();
        switch(option)
        {
            case 1:
            {
                CreateFile file = new CreateFile();
                System.out.println("Enter full path of file:");
```

```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
1
Enter full path of file:
/home/taqi/report/Files/original/rawData.txt
BUILD SUCCESSFUL (total time: 1 minute 16 seconds)
```

FIGURE 6.17: NetBeanS Screeshot of Marking Raw Dataset

Figure 6.18 is the counting how many records of raw dataset are more than 50,000 and how many of them are less than or equal 50,000.

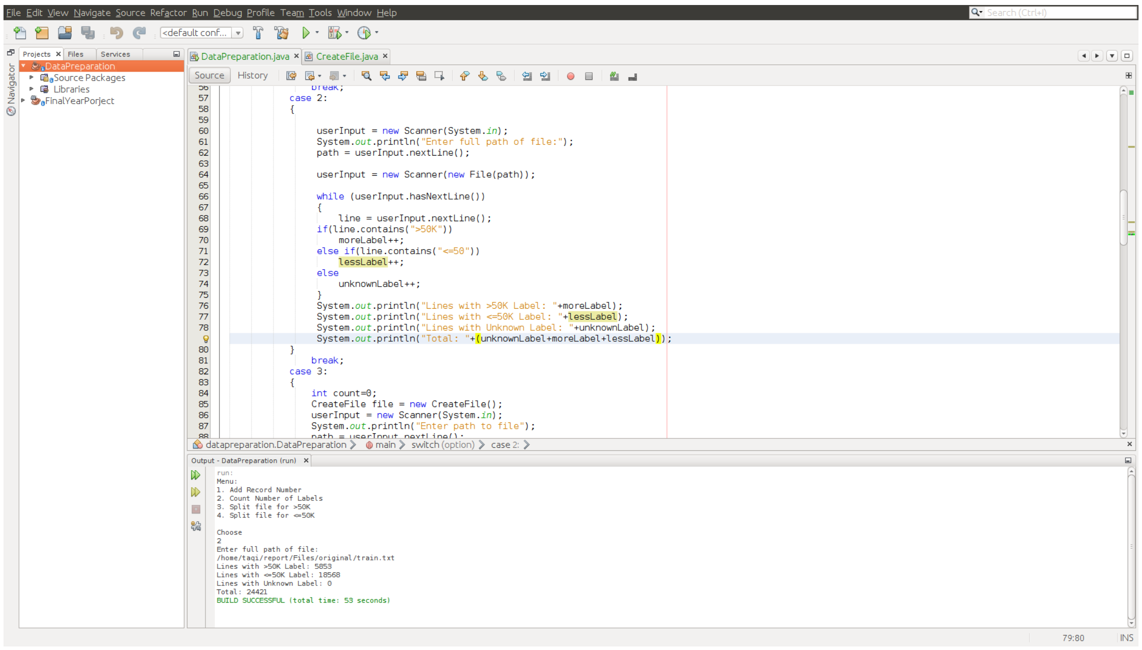


```
                break;
            case 2:
            {

                userInput = new Scanner(System.in);
                System.out.println("Enter full path of file:");
                path = userInput.nextLine();

                userInput = new Scanner(new File(path));

                while (userInput.hasNextLine())
                {
                    line = userInput.nextLine();
                if(line.contains(">50K"))
                    moreLabel++;
                else if(line.contains("<=50"))
                    lessLabel++;
                else
                    unknownLabel++;
                }
                System.out.println("Lines with >50K Label: "+moreLabel);
                System.out.println("Lines with <=50K Label: "+lessLabel);
                System.out.println("Lines with Unknown Label: "+unknownLabel);
                System.out.println("Total: "+(unknownLabel+moreLabel+lessLabel));
            }
                break;
            case 3:
            {
                int count=0;
                CreateFile file = new CreateFile();
                userInput = new Scanner(System.in);
                System.out.println("Enter path to file");
                path = userInput.nextLine();
```

```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
2
Enter full path of file:
/home/taqi/report/Files/original/rawDataNumberedFile.txt
Lines with >50K Label: 11687
Lines with <=50K Label: 37155
Lines with Unknown Label: 0
Total: 48842
BUILD SUCCESSFUL (total time: 8 seconds)
```

FIGURE 6.18: Screeshot of Counting Labels of Raw Data

Figure 6.19 is the counting how many records of train dataset are more than 50,000 and how many of them are less than or equal 50,000.

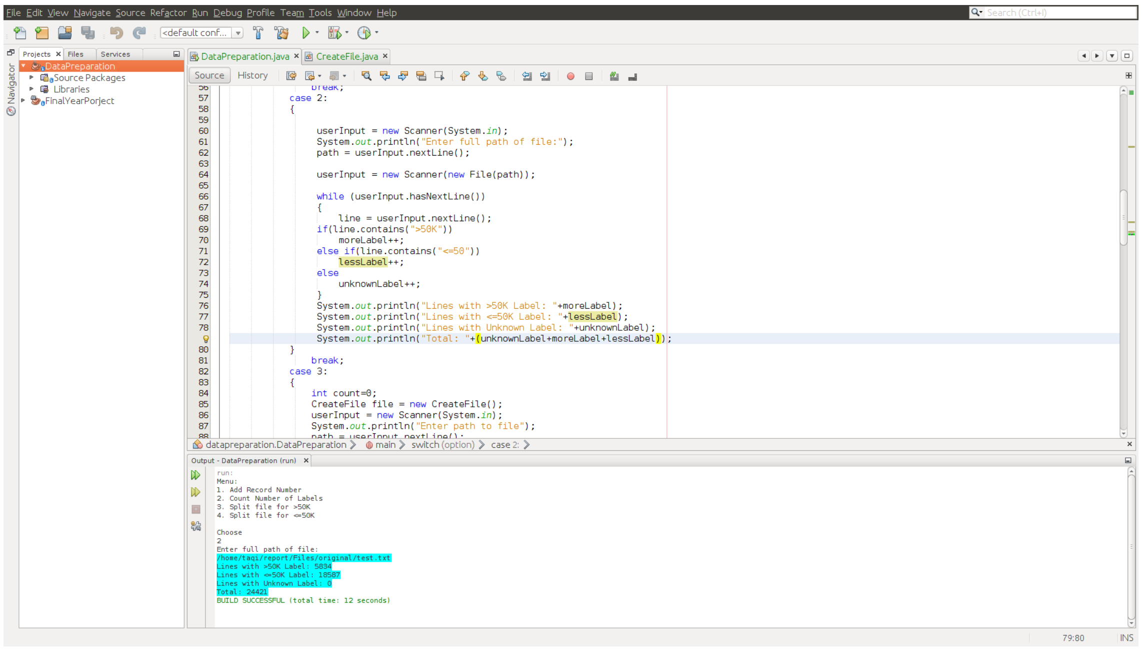


```
            case 2:
            {

                userInput = new Scanner(System.in);
                System.out.println("Enter full path of file:");
                path = userInput.nextLine();

                userInput = new Scanner(new File(path));

                while (userInput.hasNextLine())
                {
                    line = userInput.nextLine();
                if(line.contains(">50K"))
                    moreLabel++;
                else if(line.contains("<=50"))
                    lessLabel++;
                else
                    unknownLabel++;
                }
                System.out.println("Lines with >50K Label: "+moreLabel);
                System.out.println("Lines with <=50K Label: "+lessLabel);
                System.out.println("Lines with Unknown Label: "+unknownLabel);
                System.out.println("Total: "+(unknownLabel+moreLabel+lessLabel));
            }
                break;
            case 3:
            {
                int count=0;
                CreateFile file = new CreateFile();
                userInput = new Scanner(System.in);
                System.out.println("Enter path to file");
```

```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
2
Enter full path of file:
/home/taqi/report/Files/original/train.txt
Lines with >50K Label: 5853
Lines with <=50K Label: 18568
Lines with Unknown Label: 0
Total: 24421
BUILD SUCCESSFUL (total time: 53 seconds)
```

FIGURE 6.19: Screeshot Counting Labels of train.txt

Figure 6.20 is the counting how many records of test dataset are more than 50,000 and how many of them are less than or equal 50,000.

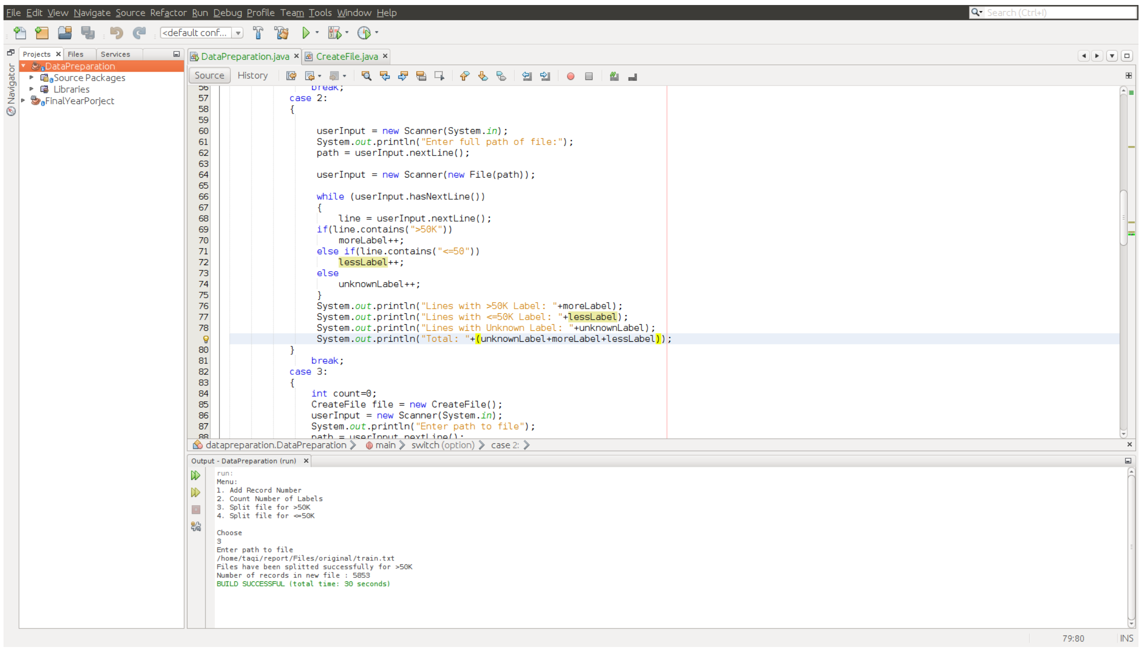


```
                break;
            case 2:
            {

                userInput = new Scanner(System.in);
                System.out.println("Enter full path of file:");
                path = userInput.nextLine();

                userInput = new Scanner(new File(path));

                while (userInput.hasNextLine())
                {
                    line = userInput.nextLine();
                if(line.contains(">50K"))
                    moreLabel++;
                else if(line.contains("<=50"))
                    lessLabel++;
                else
                    unknownLabel++;
                }
                System.out.println("Lines with >50K Label: "+moreLabel);
                System.out.println("Lines with <=50K Label: "+lessLabel);
                System.out.println("Lines with Unknown Label: "+unknownLabel);
                System.out.println("Total: "+(unknownLabel+moreLabel+lessLabel));
            }
                break;
            case 3:
            {
                int count=0;
                CreateFile file = new CreateFile();
                userInput = new Scanner(System.in);
                System.out.println("Enter path to file");
                path = userInput.nextLine();
```

```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
2
Enter full path of file:
/home/taqi/report/Files/original/test.txt
Lines with >50K Label: 5834
Lines with <=50K Label: 18587
Lines with Unknown Label: 0
Total: 24421
BUILD SUCCESSFUL (total time: 12 seconds)
```

Figure 6.20: Screeshot Counting Labels of test.txt

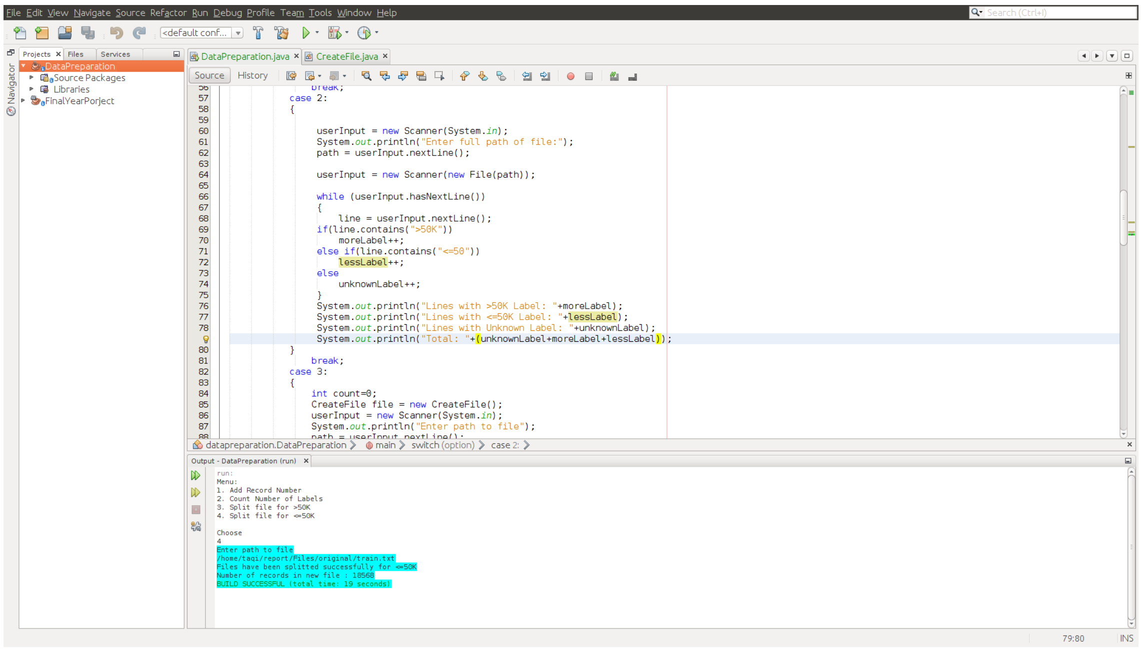
Figure 6.21 is our Java program for separating labels more than 50,000 out of train dataset.


```
                break;
            case 2:
            {

                userInput = new Scanner(System.in);
                System.out.println("Enter full path of file:");
                path = userInput.nextLine();

                userInput = new Scanner(new File(path));

                while (userInput.hasNextLine())
                {
                    line = userInput.nextLine();
                if(line.contains(">50K"))
                    moreLabel++;
                else if(line.contains("<=50"))
                    lessLabel++;
                else
                    unknownLabel++;
                }
                System.out.println("Lines with >50K Label: "+moreLabel);
                System.out.println("Lines with <=50K Label: "+lessLabel);
                System.out.println("Lines with Unknown Label: "+unknownLabel);
                System.out.println("Total: "+(unknownLabel+moreLabel+lessLabel));
            }
                break;
            case 3:
            {
                int count=0;
                CreateFile file = new CreateFile();
                userInput = new Scanner(System.in);
                System.out.println("Enter path to file");
                path = userInput.nextLine();
```

```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
3
Enter path to file
/home/taqi/report/Files/original/train.txt
Files have been splitted successfully for >50K
Number of records in new file : 5853
BUILD SUCCESSFUL (total time: 30 seconds)
```

FIGURE 6.21: Filtering Train Dataset for $> 50K$ Label

Figure 6.22 is our Java program for separating labels more than 50,000 out of train dataset.



```java
                    break;
            case 2:
            {

                userInput = new Scanner(System.in);
                System.out.println("Enter full path of file:");
                path = userInput.nextLine();

                userInput = new Scanner(new File(path));

                while (userInput.hasNextLine())
                {
                    line = userInput.nextLine();
                if(line.contains(">50K"))
                    moreLabel++;
                else if(line.contains("<=50"))
                    lessLabel++;
                else
                    unknownLabel++;
                }
                System.out.println("Lines with >50K Label: "+moreLabel);
                System.out.println("Lines with <=50K Label: "+lessLabel);
                System.out.println("Lines with Unknown Label: "+unknownLabel);
                System.out.println("Total: "+(unknownLabel+moreLabel+lessLabel));
            }
                break;
            case 3:
            {
                int count=0;
                CreateFile file = new CreateFile();
                userInput = new Scanner(System.in);
                System.out.println("Enter path to file");
                path = userInput.nextLine();
```

dataprepraration.DataPreparation > main > switch (option) > case 2: >
Output - DataPreparation (run)
79:80
INS

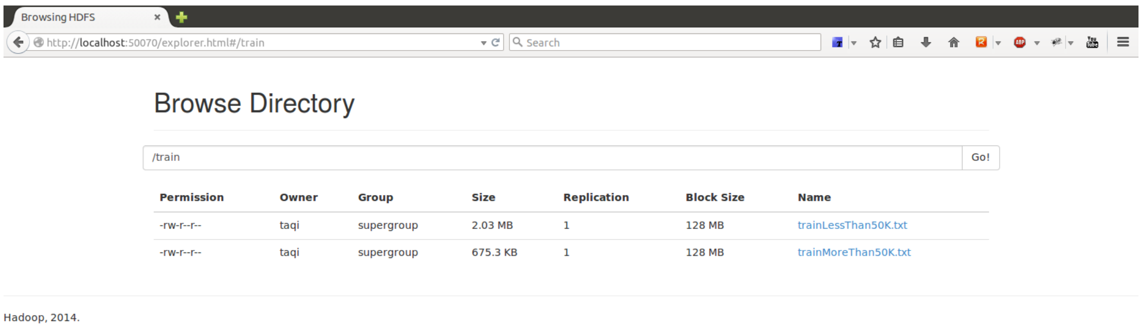
```
run:
Menu:
1. Add Record Number
2. Count Number of Labels
3. Split file for >50K
4. Split file for <=50K

Choose
4
Enter path to file
/home/taqi/report/Files/original/train.txt
Files have been splitted successfully for <=50K
Number of records in new file : 18568
BUILD SUCCESSFUL (total time: 19 seconds)
```

FIGURE 6.22: Screenshot of Filtering Train Dataset for $<= 50K$ Label

## 6.5 Classification Engine

Next few figures are the screen shot of our classification program in spark which is classifying test dataset and outputing the result of prediction for each record in file prediction.txt

First step is to copy the training dataset into Hadoop cluster so spark can read it from HDFS and the success of is shown in figure 6.23

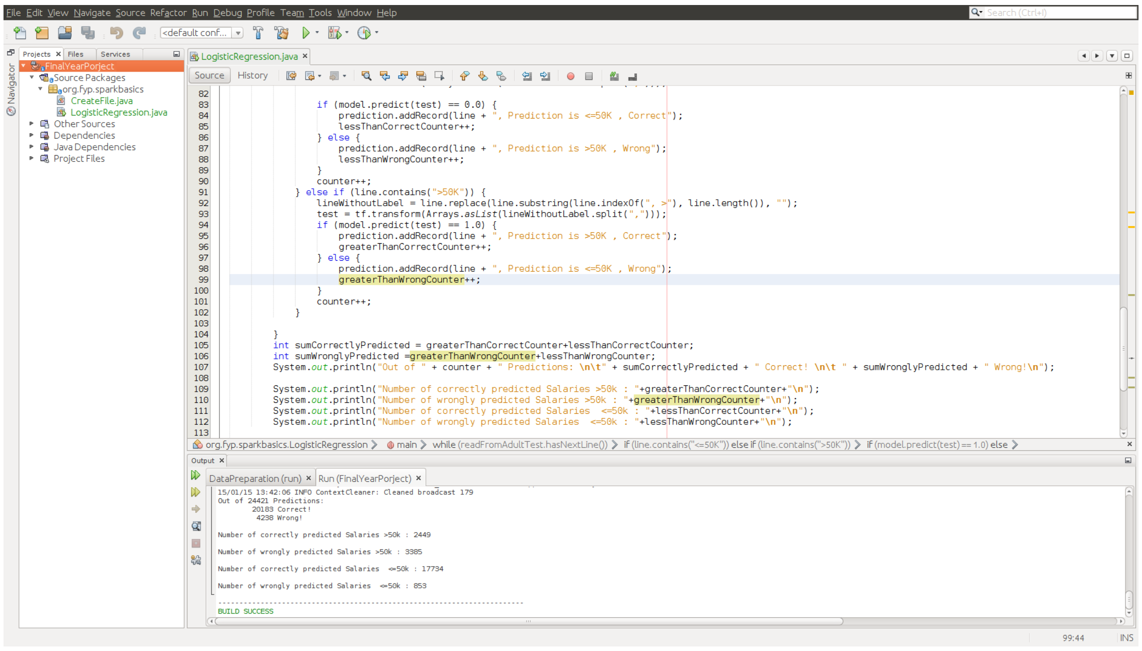


| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | taqi | supergroup | 2.03 MB | 1 | 128 MB | trainLessThan50K.txt |
| -rw-r--r-- | taqi | supergroup | 675.3 KB | 1 | 128 MB | trainMoreThan50K.txt |

Hadoop, 2014.

FIGURE 6.23: Training Dataset on HDFS

Next we meed to run our algorithms or programs in NetBeans 6.24

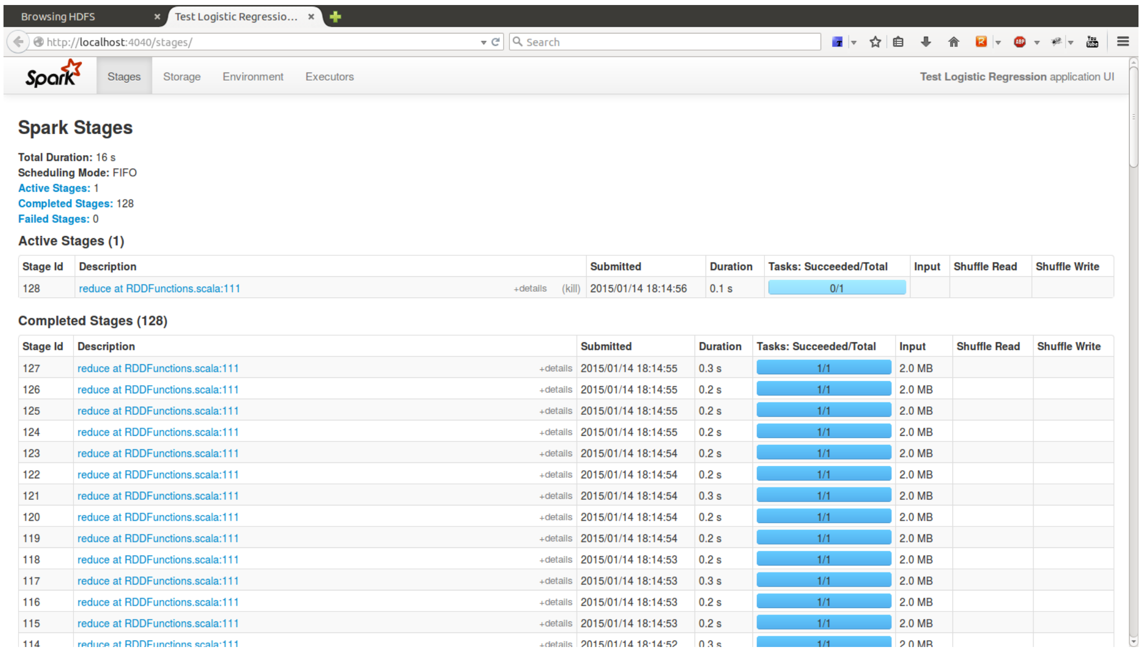


```
                if (model.predict(test) == 0.0) {
                    prediction.addRecord(line + ", Prediction is <=50K , Correct");
                    lessThanCorrectCounter++;
                } else {
                    prediction.addRecord(line + ", Prediction is >50K , Wrong");
                    lessThanWrongCounter++;
                }
                counter++;
            } else if (line.contains(">50K")) {
                lineWithoutLabel = line.replace(line.substring(line.indexOf(", >"), line.length()), "");
                test = tf.transform(Arrays.asList(lineWithoutLabel.split(",")));
                if (model.predict(test) == 1.0) {
                    prediction.addRecord(line + ", Prediction is >50K , Correct");
                    greaterThanCorrectCounter++;
                } else {
                    prediction.addRecord(line + ", Prediction is <=50K , Wrong");
                    greaterThanWrongCounter++;
                }
                counter++;
            }

        }
        int sumCorrectlyPredicted = greaterThanCorrectCounter+lessThanCorrectCounter;
        int sumWronglyPredicted =greaterThanWrongCounter+lessThanWrongCounter;
        System.out.println("Out of " + counter + " Predictions: \n\t" + sumCorrectlyPredicted + " Correct! \n\t " + sumWronglyPredicted + " Wrong!\n");

        System.out.println("Number of correctly predicted Salaries >50k : "+greaterThanCorrectCounter+"\n");
        System.out.println("Number of wrongly predicted Salaries >50k : "+greaterThanWrongCounter+"\n");
        System.out.println("Number of correctly predicted Salaries  <=50k : "+lessThanCorrectCounter+"\n");
        System.out.println("Number of wrongly predicted Salaries  <=50k : "+lessThanWrongCounter+"\n");
```

```
15/01/15 13:42:06 INFO ContextCleaner: Cleaned broadcast 179
Out of 24421 Predictions:
        20183 Correct!
         4238 Wrong!

Number of correctly predicted Salaries >50k : 2449

Number of wrongly predicted Salaries >50k : 3385

Number of correctly predicted Salaries  <=50k : 17734

Number of wrongly predicted Salaries  <=50k : 853

------------------------------------------------------------------------
BUILD SUCCESS
```

FIGURE 6.24: Screenshot of Netbeans of Classifying program

Once the classification is running we can see the progress of it on web UI. Figure 6.25 and figure 6.26 are showing that program is running on the background and it is busy with classification of test dataset.


Browsing HDFS
Test Logistic Regressio...
http://localhost:4040/stages/
Search
Spark
Stages
Storage
Environment
Executors
Test Logistic Regression application UI

## Spark Stages

**Total Duration:** 16 s
**Scheduling Mode:** FIFO
**Active Stages:** 1
**Completed Stages:** 128
**Failed Stages:** 0

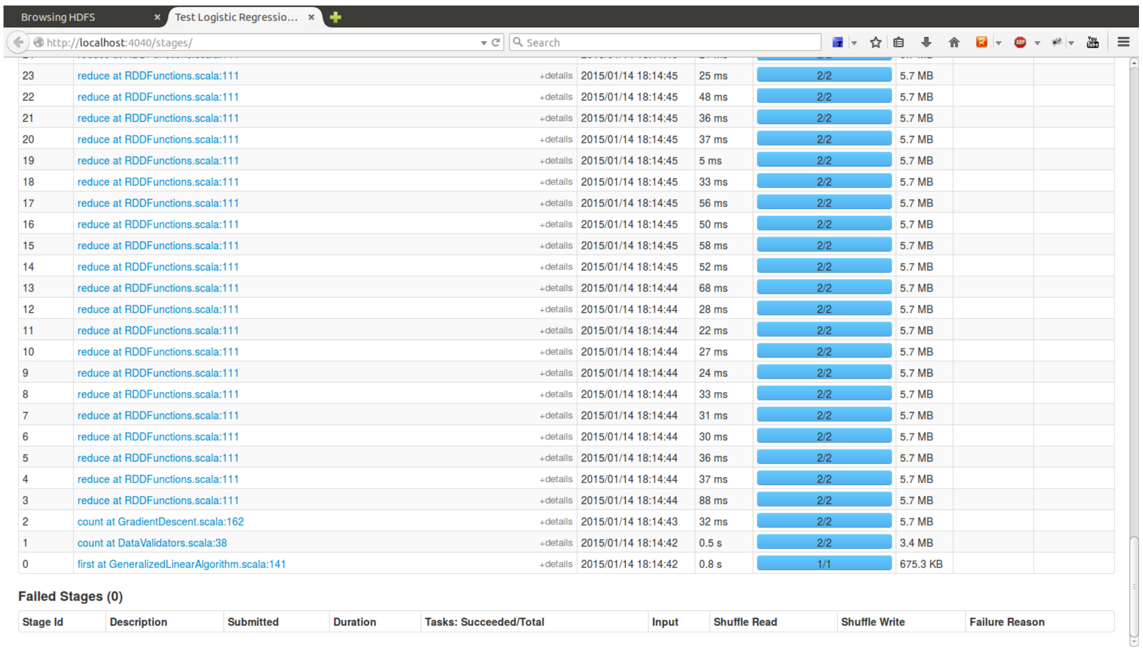
**Active Stages (1)**

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| 128 | reduce at RDDFunctions.scala:111 +details (kill) | 2015/01/14 18:14:56 | 0.1 s | 0/1 | | | |

**Completed Stages (128)**

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| 127 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:55 | 0.3 s | 1/1 | 2.0 MB | | |
| 126 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:55 | 0.2 s | 1/1 | 2.0 MB | | |
| 125 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:55 | 0.2 s | 1/1 | 2.0 MB | | |
| 124 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:55 | 0.2 s | 1/1 | 2.0 MB | | |
| 123 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:54 | 0.2 s | 1/1 | 2.0 MB | | |
| 122 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:54 | 0.2 s | 1/1 | 2.0 MB | | |
| 121 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:54 | 0.3 s | 1/1 | 2.0 MB | | |
| 120 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:54 | 0.2 s | 1/1 | 2.0 MB | | |
| 119 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:54 | 0.2 s | 1/1 | 2.0 MB | | |
| 118 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:53 | 0.2 s | 1/1 | 2.0 MB | | |
| 117 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:53 | 0.3 s | 1/1 | 2.0 MB | | |
| 116 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:53 | 0.2 s | 1/1 | 2.0 MB | | |
| 115 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:53 | 0.2 s | 1/1 | 2.0 MB | | |
| 114 | reduce at RDDFunctions.scala:111 +details | 2015/01/14 18:14:52 | 0.3 s | 1/1 | 2.0 MB | | |

FIGURE 6.25: Spark Web UI Running Classification



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 23 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 25 ms | 2/2 | 5.7 MB | | |
| 22 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 48 ms | 2/2 | 5.7 MB | | |
| 21 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 36 ms | 2/2 | 5.7 MB | | |
| 20 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 37 ms | 2/2 | 5.7 MB | | |
| 19 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 5 ms | 2/2 | 5.7 MB | | |
| 18 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 33 ms | 2/2 | 5.7 MB | | |
| 17 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 56 ms | 2/2 | 5.7 MB | | |
| 16 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 50 ms | 2/2 | 5.7 MB | | |
| 15 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 58 ms | 2/2 | 5.7 MB | | |
| 14 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:45 | 52 ms | 2/2 | 5.7 MB | | |
| 13 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 68 ms | 2/2 | 5.7 MB | | |
| 12 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 28 ms | 2/2 | 5.7 MB | | |
| 11 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 22 ms | 2/2 | 5.7 MB | | |
| 10 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 27 ms | 2/2 | 5.7 MB | | |
| 9 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 24 ms | 2/2 | 5.7 MB | | |
| 8 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 33 ms | 2/2 | 5.7 MB | | |
| 7 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 31 ms | 2/2 | 5.7 MB | | |
| 6 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 30 ms | 2/2 | 5.7 MB | | |
| 5 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 36 ms | 2/2 | 5.7 MB | | |
| 4 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 37 ms | 2/2 | 5.7 MB | | |
| 3 | reduce at RDDFunctions.scala:111 | +details | 2015/01/14 18:14:44 | 88 ms | 2/2 | 5.7 MB | | |
| 2 | count at GradientDescent.scala:162 | +details | 2015/01/14 18:14:43 | 32 ms | 2/2 | 5.7 MB | | |
| 1 | count at DataValidators.scala:38 | +details | 2015/01/14 18:14:42 | 0.5 s | 2/2 | 3.4 MB | | |
| 0 | first at GeneralizedLinearAlgorithm.scala:141 | +details | 2015/01/14 18:14:42 | 0.8 s | 1/1 | 675.3 KB | | |

**Failed Stages (0)**

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Shuffle Read | Shuffle Write | Failure Reason |
|---|---|---|---|---|---|---|---|---|

FIGURE 6.26: Spark Web UI Running Classification-2

## 6.6 Adult Dataset Accounting

| **File Name** | **No. of Records Unknown/Unknown** | $> 50K$ | $<= 50K$ |
|---|---|---|---|
| rawData.txt | 48842 / 45222 | 11687 | 37155 |
| rawDataNumberedFile.txt | 48842 / 45222 | 11687 | 37155 |
| train.txt | 24421 /22611 | 5853 | 18568 |
| trainLessThan50K.txt | 18568 | 0 | 18568 |
| trainMoreThan50K.txt | 5853 | 5853 | 0 |
| test.txt | 24421 / 22611 | 5834 | 18587 |

TABLE 6.1: Adult Dataset Accounting

## 6.7 Findings

Based on the above Tanle of result and after running the program to classify the elements of test.txt before and after removing the unknown elements we got the following result.

Out of 24421 Predictions: 20183 Correct! 4238 Wrong!

Number of correctly predicted Salaries $> 50k$ : 2449

Number of wrongly predicted Salaries $> 50k$ : 3385

Total 5834

Number of correctly predicted Salaries $<= 50k$ : 17734

Number of wrongly predicted Salaries $<= 50k$ : 853

Total 18587

For classes above $> 50K$ we correctly predicted 41.97% of data and could which is very low accuracy but for the classes $<= 50K$ we correctly predicted 95.41 % of the data which is very high accuracy.

The reason for getting a very high error rate in set of $> 50K$ might be because of the distribution of values in their features. In the set of people who make more than 50K there is a wide variation that cause the inaccurate prediction. While in individuals who make less than or equal 50K these variation is less.

Another justification for this outcome of the classifier is that because the number of classes less than or equal label 50K is more than number of classes greater than 50K the algorithm had learned better and it could predicted with high accuracy.

In order to get better result in large-scale programming we need a very large data and due to machine limitation we could not run the algorithm against the Big

Data.

Another approach can be reversing the process of training the function of logistic regression where it see one label based on the feature and throw the other without checking into another category.

However the point of doing this prediction and running and writing this algorithm is learning and now that we know how Big Companies such as Google, Amazon, Facebook, Yahoo!, etc. use machine learning algorithms to serve their customer or get and understanding of their customer behavior we can claim that maybe not fully but partly we also reached our goal.

# Chapter 7

# Conclusion

For phase one of this project we have reviewed number of books and writings as well as documentations of Apache Hadoop and Spark to reach the goal we have set in for the first phase in the beginning of our report.
Objective for the phase one were:

1. Understand Big Data and Big Data Processing
2. Understand What Hadoop is and how the main components of Hadoop (MapReudce and HDFS) work together
3. Learn the drawbacks of Hadoop and the reason to use Spark
4. Wrote a simple program to run on Hadoop

What we have achieved by now:

- We did look at what Big Data is and what are the current challenges of Big Data. We understand how Hadoop has been developed and the reasons of it.
- We took a look at two main component of Hadoop, MapReduce and HFDS
- We have learned how to set up Hadoop cluster on single node machine as well multi node cluster.
- We examined how Hadoop works and how we can run a MapReduce application to get the data out of HDFS and how to store data in our HDFS
- We successfully installed Spark on out clusterrocks machines on FCI Lab.

For the second phase of project we have learned a lot of things and faced with many problems. Scope for the phase two were:

1. Identifying a real world problem to solve with machine learning

2. Using Spark and Mllib to overcome the problems of MapReduce
3. Understanding of machine learning concepts in industry and real life
4. Write a program for to solve the problem that we identified

What we have achieved by end of this project:

- We were able to increase our understanding of Hadoop from the first phase to better do work with Spark.
- We indeed found problem of predicting salary prediction for set of individuals
- We learned the data analysis and machine learning concepts and different classification algorithms
- We learned about Spark and its architecture and it Machine learning library
- Using the API of Spark we wrote classification program that could classify our dataset of salaries based on its feature and by using logistic regression

## 7.1 What Else We Have Learned!

### 7.1.1 COMMUNICATION

1. **Google Group**
   The main mode of communication for this FYP was via Google Group. We have set up a private Google Group for all FYP students in for the purpose of sharing knowledge and material we gain through this FYP. It is not publicly visible or accessible.

2. **Google Group URL**
   `https://groups.google.com/forum/?hl=en#!forum/fyp_mmu2011`

3. **Google Group Email**
   fyp_mmu2011@googlegroups.com

4. **Membership by invitation**
   You have to be invited to a become member of this group. To be a member, you must have a **gmail** account (Note: GEEE mail).

5. **How to get invited**
   Write a gmail message to ***wruslan.hahaha@gmail.com*** with the subject *"Request to Join FYP_MMU2011 Google Group"* and provide your gmail address.

### 7.1.2 LaTex

I am very thankful to Mr.Ruslan for requiring me to learn and write my report in LaTeXfrom the beginning of this project.

LaTex is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents. Being able to produce well-structured document will surely help me in my future jobs and studies. I hope I can write my own book using LaTeXabout large-scale programming.

### 7.1.3 Git

Git is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear work-flows. Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since become the most widely adopted version control system for software development.[13]

Beside Google group we have also set up a git repository on our clusterrocks to be able to see what changes who made in the files and in this way supervisor have more control over what student will go through. How is the progress and whether the student in on the right path or not.

Being able to work with Git will help me participating on big software development with large number of developers around the globe through services such as `Github.com`

### 7.1.4 Follow Plan, Start Early

There is a saying, *"If you fail to plan, you plan to fail"*, and I truly understand the meaning of this expression during this final year project.

It happened a lot that I decided to do some task but since I did not have appropriate plan to do it I could not finish it on time. However one of the good things I learned was planning before starting something.

Another good lesson I learned is *"if you want to finish early you gotta start early"*.

# Bibliography

[1] Gary Turkington. *Hadoop-Beginner's Guide.* Packt Publishing Ltd, Livery Place 35 Livery Street Birmingham B3 2PB, UK., 2013.

[2] Lewis John Mcgibbney. Welcome to the apache nutch wiki. `http://wiki.apache.org/nutch/`, 2014. [Online; accessed 17-August-2014].

[3] Lucene. `http://en.wikipedia.org/wiki/Lucene`, 2014. [Online; accessed 17-August-2014].

[4] Alexandru Adrian TOLE. Big data challenges.

[5] WENMING QIU YINGWEI JIN YUJIE XU UGHEGHUKWU AWADA KEQIU LI CHANGQING JI, YULI. Big data processing: Big challenges and opportunities. 2012.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. 2004.

[7] Apache spark. `http://en.wikipedia.org/wiki/Apache_Spark`, 2014. [Online; accessed 12-January-2015].

[8] Spark overview. `http://spark.apache.org/`, 2014. [Online; accessed 20-August-2014].

[9] Peter Harrington. *Machine Learning in Action.* Manning, Manning Publications Co. 180 Broad St.Suite 1323 Stamford, CT 06901, 2012.

[10] Spark mllib page. http://spark.apache.org/mllib, 2014. [Online; accessed 12-January-2015].

[11] Logistic regression in apache spark. https://samarthbhargav.wordpress.com/2014/04/22/logistic-regression-in-apache-spark/, 2014. [Online; accessed 12-January-2015].

[12] Setting up a single node cluster. http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/SingleCluster.html, 2014. [Online; accessed 20-August-2014].

[13] Git (software). http://en.wikipedia.org/wiki/Git_(software), 2014. [Online; accessed 12-January-2015].

# Appendix A

# Hadoop Single Node Setup

In this Appendix we explain how to install Hadoop on single machine which is basically our laptop.

## Bash Scripts - Hadoop Prerequisites Installations

In order to be able to install Hadoop on your computer you need to have the following prerequisites:

1. **JDK 6+**: Since Hadoop is written Java is primary software that is needed to run Hadoop.
2. **SSH (Secure Shell)**: is a secure shell that usually runs on top of SSL and has a built-in username/password authentication scheme that can be used for secure access to a remote host; it is a more secure alternative to rlogin and telnet.

It is a good idea to use SSH for remote administration purposes (instead of rlogin, for example). But note that it is not used to secure communication among the elements in a Hadoop cluster (DataNode, NameNode, TaskTracker or YARN ResourceManager, JobTracker or YARN NodeManager, or the /etc/init.d scripts that start daemons locally).

The Hadoop components use SSH in the following cases:

The sshfencer component of High Availability Hadoop configurations uses SSH; the shell fencing method does not require SSH.
Whirr uses SSH to enable secure communication with the Whirr cluster in the Cloud.

3. **Linux Operating System**: Although Hadoop can be installed on any operating systems, we recommend installing Linux for better performance and less work. Our Linux distribution for this writing was Ubuntu 14.04

4. **rsync**:rsync is a widely-used utility to keep copies of a file on two computer systems the same which is what we do in cluster setup. Later for the cluster setup we need *rsync* but we will show all scripts for the prerequisites here:

```
Script started on Wed 20 Aug 2014 05:09:14 PM MYT
taqi@taqi:~$ sudo apt-get install openjdk-7-jdk
taqi@taqi:~$ java -version
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.1) (7u65-2.5.1-4ubuntu1~0.14.04.1)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)

taqi@taqi:~$ sudo apt-get install openssh-server
taqi@taqi:~$ sudo apt-get install ssh
taqi@taqi:~$ sudo apt-get install rsync
```

## Bash Scripts - Hadoop Installation

In order to have Hadoop up and running there are some steps that needs to be taken and we have put them all in the following set of scripts:

```
==========================================
Step 1: Download hadoop-2.4.1 from apache website
==========================================

taqi@taqi:~$ cd Downloads/
taqi@taqi:~/Downloads$ ls |grep hadoop
hadoop-2.4.1.tar.gz <--- Our Downloaded File

==========================================
Step 2: Extract compressed file into /usr/local
==========================================

taqi@taqi:~/Downloads$ tar vxzf hadoop-2.4.1.tar.gz -C /usr/local

==========================================
Step 3: Check if you have them in your /usr/local
==========================================
taqi@taqi:~/Downloads$ cd /usr/local/
taqi@taqi:/usr/local$ ll
total 44
drwxr-xr-x 11 root root 4096 Aug 20 04:23 ./
drwxr-xr-x 10 root root 4096 Jul 23 05:57 ../
drwxr-xr-x  2 root root 4096 Aug 20 17:02 bin/
drwxr-xr-x  2 root root 4096 Jul 23 05:57 etc/
drwxr-xr-x  2 root root 4096 Jul 23 05:57 games/
drwxr-xr-x 13 taqi taqi 4096 Aug 22 03:19 hadoop-2.4.1/ <--- Our Extracted File
drwxr-xr-x  2 root root 4096 Jul 23 05:57 include/
drwxr-xr-x  5 root root 4096 Aug 18 20:14 lib/
lrwxrwxrwx  1 root root    9 Aug 18 18:15 man -> share/man/
drwxr-xr-x  2 root root 4096 Jul 23 05:57 sbin/
drwxr-xr-x  9 root root 4096 Aug 20 17:02 share/
drwxr-xr-x  2 root root 4096 Jul 23 05:57 src/
==========================================
Step 4: Change to Hadoop directory
==========================================
taqi@taqi:~$ cd ./hadoop-2.4.1/
taqi@taqi:/usr/local/hadoop-2.4.1$ ll
total 76
drwxr-xr-x 13 taqi taqi  4096 Aug 22 03:19 ./
drwxr-xr-x 11 root root  4096 Aug 20 04:23 ../
drwxr-xr-x  2 taqi taqi  4096 Jun 21 14:05 bin/
```

```
drwxr-xr-x  3 taqi taqi  4096 Jun 21 14:05 etc/
drwxr-xr-x  2 taqi taqi  4096 Jun 21 14:05 include/
drwxrwxr-x  2 taqi taqi  4096 Aug 19 23:59 input/
drwxr-xr-x  3 taqi taqi  4096 Jun 21 14:05 lib/
drwxr-xr-x  2 taqi taqi  4096 Jun 21 14:05 libexec/
-rw-r--r--  1 taqi taqi 15458 Jun 21 14:38 LICENSE.txt
drwxrwxr-x  3 taqi taqi  4096 Aug 25 14:13 logs/
-rw-r--r--  1 taqi taqi   101 Jun 21 14:38 NOTICE.txt
drwxrwxr-x  3 taqi taqi  4096 Aug 20 00:18 output/
-rw-r--r--  1 taqi taqi  1366 Jun 21 14:38 README.txt
drwxr-xr-x  2 taqi taqi  4096 Jun 21 14:05 sbin/
drwxr-xr-x  4 taqi taqi  4096 Jun 21 14:05 share/
drwxrwxr-x  2 taqi taqi  4096 Aug 22 04:01 wordcount_classes/
=================================================
Step 5: Run the following to make sure you have correctly installed Hadoop
=================================================
taqi@taqi:/usr/local/hadoop-2.4.1$ bin/hadoop version
Hadoop 2.4.1
Subversion http://svn.apache.org/repos/asf/hadoop/common -r 1604318
Compiled by jenkins on 2014-06-21T05:43Z
Compiled with protoc 2.5.0
From source with checksum bb7ac0a3c73dc131f4844b873c74b630
This command was run using
  /usr/local/hadoop-2.4.1/share/hadoop/common/hadoop-common-2.4.1.jar
```

## Bash Scripts - Environmental Settings

Once you have installed Hadoop if you want to be able to run Hadoop from any directory that you are in terminal you need to add the following environment variables to point to your Hadoop directory, otherwise for each time of running Hadoop you need to go to the bin directory of your Hadoop folder. The following bash scripts are optional to Hadoop installation and it is more of personal choice.

```
taqi@taqi:~$ sudo gedit ~/.bashrc
[sudo] password for taqi:
==========================================
Type the following in the opened window
==========================================


export JAVA_HOME=/usr/lib/jvm/jdk/
export HADOOP_INSTALL=/usr/local/hadoop-2.4.1
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
```

## Bash Scripts Running Hadoop on Single Node Machine

If you are running Hadoop for the first time on your single machine you need to format the namenode before running Hadoop daemons. Once you have successful format message you can run your Hadoop.
The following scripts shows how to run Hadoop for the first time.

```
Script started on Mon 25 Aug 2014 02:12:16 PM MYT
taqi@taqi:~$ hdfs namenode -format
14/08/25 14:12:30 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = taqi/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.4.1
STARTUP_MSG:   classpath = /usr/local/hadoop-2.4.1/etc/hadoop:
...
...
...
14/08/25 14:12:31 INFO namenode.FSImage:
 Allocated new BlockPoolId: BP-432914324-127.0.1.1-1408947151658
14/08/25 14:12:31 INFO common.Storage:
Storage directory /tmp/hadoop-taqi/dfs/name has been successfully formatted. <--
    SUCCESS MESSAGE
14/08/25 14:12:31 INFO namenode.NNStorageRetentionManager:
Going to retain 1 images with txid >= 0
14/08/25 14:12:31 INFO util.ExitUtil: Exiting with status 0
14/08/25 14:12:31 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at taqi/127.0.1.1
************************************************************/
```

```
ONCE YOU GET THE SUCCESS MESSAGE YOU ARE GOOD
TO GO TO THIS STEP RUNNING HADOOP DEAMONS
==============================================

taqi@taqi:~$ start-dfs.sh
localhost: starting datanode,
logging to /usr/local/hadoop-2.4.1/logs/hadoop-taqi-datanode-taqi.out
Starting secondary namenodes
0.0.0.0: starting secondarynamenode,
logging to /usr/local/hadoop-2.4.1/logs/hadoop-taqi-secondarynamenode-taqi.out

taqi@taqi:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager,
logging to /usr/local/hadoop-2.4.1/logs/yarn-taqi-resourcemanager-taqi.out
localhost: starting nodemanager,
logging to /usr/local/hadoop-2.4.1/logs/yarn-taqi-nodemanager-taqi.out

==============================================
CHECK IF YOU HAVE ALL FIVE RUNNING DEAMONS UP AND RUNNING
==============================================

taqi@taqi:~$ jps
6356 ResourceManager
6200 SecondaryNameNode
6783 Jps
6488 NodeManager
5770 NameNode
5922 DataNode
```

# Appendix B

# Hadoop Cluster Setup

Installing a Hadoop cluster involves :

1. First of all in order to Install Hadoop on cluster mode you need to have root privileges to be able to copy, sync, change content of some files, and install required software on your cluster if needed.

```
taqi@taqi-laptop:~$ ssh taqi@clusterrocks
Password:
Last login: Wed Jan 21 09:48:56 2015 from 10.106.17.220
Rocks 6.1 (Emerald Boa)
Profile built 10:52 14-Feb-2014

Kickstarted 19:47 14-Feb-2014
[taqi@clusterrocks ~]$ sudo -i
[sudo] password for taqi:
[root@clusterrocks ~]#
```

2. Downloading a stable version of Hadoop from Apache mirrors (`http://hadoop.apache.org/releases.html#Download`) and copy it into master node of your cluster

```
taqi@taqi-laptop:~$ wget http://supergsego.com/apache/hadoop/common/stable/
    hadoop-2.6.0.tar.gz
--2015-01-26 22:03:42--  http://supergsego.com/apache/hadoop/common/stable/
    hadoop-2.6.0.tar.gz
Resolving supergsego.com (supergsego.com)... 173.12.119.133
Connecting to supergsego.com (supergsego.com)|173.12.119.133|:80...
    connected.
HTTP request sent, awaiting response... 200 OK
Length: 195257604 (186M) [application/x-gzip]
Saving to: 'hadoop-2.6.0.tar.gz'
100%[=========================================>] 195,257,604  231KB/s   in 10m
     0s

2015-01-26 22:13:42 (318 KB/s) - 'hadoop-2.6.0.tar.gz' saved
    [195257604/195257604]
==================================================
Make sure you have successfully download the Hadoop
==================================================
taqi@taqi-laptop:~$ ls -al |grep "hadoop"
-rw-rw-r-- 1 taqi taqi 195257604 Dec  1 07:52 hadoop-2.6.0.tar.gz

==================================================
Copy downloaded files into your cluster master node
==================================================
taqi@taqi-laptop:~$ scp hadoop-2.6.0.tar.gz taqi@clusterrocks:/home/taqi/
    Downlod/
Password:
hadoop-2.6.0.tar.gz                          100%  186MB  11.6MB/s   00:16

==================================================
Login to your ssh account and see if you have
successfully copied Hadoop files into cluster
===================================================
taqi@taqi-laptop:~$ ssh taqi@clusterrocks
Password:
Last login: Mon Jan 26 21:42:02 2015 from 10.100.213.178
Rocks 6.1 (Emerald Boa)
Profile built 10:52 14-Feb-2014

Kickstarted 19:47 14-Feb-2014
[taqi@clusterrocks ~]$ cd /home/taqi/Download/
[taqi@clusterrocks Download]$ ls -al |grep "hadoop"
-rw-rw-r-- 1 taqi taqi 195257604 Jan 26 23:11 hadoop-2.6.0.tar.gz
```

3. Unpack the software to all machines in the cluster directory you want to install Hadoop on. For our case we want to install it on /usr/local/ directory.

```
[taqi@clusterrocks Download]$ tar xvf hadoop-2.6.0.tar.gz
[taqi@clusterrocks Download]$ ls -al |grep hadoop
drwxr-xr-x 9 taqi taqi      4096 Nov 14 05:20 hadoop-2.6.0
-rw-rw-r-- 1 taqi taqi 195257604 Jan 26 23:11 hadoop-2.6.0.tar.gz
==================================================
After extracting copy the folder into /usr/local/
==================================================
[taqi@clusterrocks Download]$ sudo mv hadoop-2.6.0 /usr/local/
[sudo] password for taqi:

[taqi@clusterrocks local]$ ll | grep hadoop
drwxrwxrwx 11 root  root  4096 Feb 24  2014 hadoop-2.2.0
drwxrwxrwx 11 root  root  4096 Jan  8 20:42 hadoop-2.4.1
drwxrwxrwx 10 root  root  4096 Aug 22 20:52 hadoop-2.5.0
drwxr-xr-x  9 taqi  taqi  4096 Nov 14 05:20 hadoop-2.6.0
drwxrwxrwx 11 root  root  4096 Jan 19 15:00 spark-1.0.2-bin-hadoop2
drwxrwxrwx 11  1000 1000  4096 Jan 19 18:36 spark-1.1.1-bin-hadoop2.4
======================================================
Change mode to 777 to give rights to other to be able to run hadoop
======================================================

[taqi@clusterrocks local]$ sudo chmod -R 777 hadoop-2.6.0

[taqi@clusterrocks local]$ ll | grep hadoop
drwxrwxrwx 11 root  root  4096 Feb 24  2014 hadoop-2.2.0
drwxrwxrwx 11 root  root  4096 Jan  8 20:42 hadoop-2.4.1
drwxrwxrwx 10 root  root  4096 Aug 22 20:52 hadoop-2.5.0
drwxrwxrwx  9 taqi  taqi  4096 Nov 14 05:20 hadoop-2.6.0
drwxrwxrwx 11 root  root  4096 Jan 19 15:00 spark-1.0.2-bin-hadoop2
drwxrwxrwx 11  1000 1000  4096 Jan 19 18:36 spark-1.1.1-bin-hadoop2.4
```

4. Typically one machine in the cluster is designated as the NameNode and another machine the as ResourceManager, exclusively. These are the masters. However due to machine limitations for our case we use one machine for both purposes, our master node will be ResourceManager and NameNode at the same time for all other machines.
   In order to set the configuration files you need to tell Hadoop what are the slave machines and who is the master; so, follow the following steps to do it:

```
==================================
1. go the the configuration directory
==================================
[taqi@clusterrocks \~{}]\$ cd /usr/local/hadoop-2.6.0/etc/hadoop/

==================================
2. You need the name of hosts for datanode and namenode
which is normaly is saved on hosts file in /etc/ directory
perform the following command to get to know the name of hosts
==================================
[taqi@clusterrocks hadoop]\$ cat /etc/hosts
\# Added by rocks report host \#
\#        DO NOT MODIFY        \#
\# Add any modifications to    \#
\#    /etc/hosts.local file    \#

127.0.0.1 localhost.localdomain localhost

10.1.1.1     clusterrocks.local   clusterrocks
10.1.255.254  compute-0-0.local compute-0-0
....
UP TO
10.1.255.245  compute-0-9.local compute-0-9
10.106.17.102 clusterrocks.mmu.edu.my
======================================
3. Copy the name of the nodes that you want to use
for the Hadoop on file called "slaves" which is on
======================================
[taqi@clusterrocks hadoop]\$ sudo nano slaves
[taqi@clusterrocks hadoop]\$ cat slaves
clusterrocks
compute-0-0
....
UP TO
compute-0-9
```

```
==========================================
Login as root you need to make sure master can access slaves via ssh pass-
    less
=========================================
[taqi@clusterrocks hadoop]# sudo -i
[root@clusterrocks hadoop]# updatedb
```

5. Environmental Settings: In order to run and stop Hadoop on Clusterrocks we create a shell scripts to do put all necessary environmental variables on it and also be able to put scripts of running and stopping Hadoop in those files.
   We call them *start-hadoop-clusterrocks.sh* and *stop-hadoop-clusterrocks.sh*

   The following Scripts shows content of stat-hadoop-clusterrocks.sh:

```
[root@clusterrocks hadoop-2.6.0]# cat start-hadoop-clusterrocks.sh
#! /bin/bash

# File: start-hadoop-clusterrocks.sh
# Date: Thu Feb 20 23:08:52 MYT 2014
# Author: WRY

# OPENING MESSAGE
echo
echo "BISMILLAH. STARTING HADOOP ON CLUSTERROCKS."
echo

# PROVIDE ACTUAL SITE SETTINGS

export HADOOP_CLUSTER_NAME=clusterrocks

export JAVA_HOME=/usr/java/latest
export JSVC_HOME=/usr/java/latest

export HADOOP_PREFIX=/usr/local/hadoop-2.6.0
export HADOOP_CONF_DIR=/usr/local/hadoop-2.6.0/etc/hadoop
export HADOOP_YARN_HOME=/usr/local/hadoop-2.6.0

export HADOOP_HOME=/usr/local/hadoop-2.6.0
export HADOOP_MAPRED_HOME=/usr/local/hadoop-2.6.0
export HADOOP_COMMON_HOME=/usr/local/hadoop-2.6.0
export HADOOP_HDFS_HOME=/usr/local/hadoop-2.6.0

# EXECUTE COMMAND
```

```
$HADOOP_PREFIX/bin/hadoop version
echo
java -version
echo

# DISPLAY ENVIRONMENT VARIABLES
echo "CLUSTER_NAME      = " $HADOOP_CLUSTER_NAME
echo "JAVA_HOME         = " $JAVA_HOME
echo "JSVC_HOME         = " $JSVC_HOME
echo "HADOOP_PREFIX     = " $HADOOP_PREFIX
echo "HADOOP_CONF_DIR   = " $HADOOP_CONF_DIR
echo "HADOOP_YARN_HOME = " $HADOOP_YARN_HOME

# START hdfs ONLY ONCE TO FORMAT HADOOP FILE SYSTEM SERVICES
# OTHERWISE RE-FORMAT HDFS.

# $HADOOP_PREFIX/bin/hdfs namenode -format $HADOOP_CLUSTER_NAME

echo
echo "STARTING FIVE(5) REQUIRED HADOOP SERVICES"
echo "=============================================="

echo
echo "Executing 1/5 ... RUN ON MASTER"
  $HADOOP_PREFIX/sbin/hadoop-daemon.sh --config $HADOOP_CONF_DIR --script
     hdfs start namenode

# ====> NOTE: For datanode and nodemanager, scripts are *-daemons.sh and
     not *-daemon.sh.
# ====> Because daemon.sh does not lookup in slaves file and hence, will
     only start processes on master.

echo
echo "Executing 2/5 ... RUN ON MASTER AND SLAVES"
  $HADOOP_PREFIX/sbin/hadoop-daemons.sh  --config $HADOOP_CONF_DIR --script
      hdfs start datanode

echo
echo "Executing 3/5 ... RUN ON MASTER"
  $HADOOP_YARN_HOME/sbin/yarn-daemon.sh --config $HADOOP_CONF_DIR start
     resourcemanager

echo
echo "Executing 4/5 ... RUN ON MASTER AND SLAVES"
  $HADOOP_YARN_HOME/sbin/yarn-daemons.sh --config $HADOOP_CONF_DIR start
     nodemanager

echo
```

```
echo "Executing 5/5 ... RUN ON MASTER"
  $HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh start historyserver --config
    $HADOOP_CONF_DIR

# CLOSING MESSAGE
echo
echo "ALHAMDULILLAH. SUCCESS STARTING HADOOP ON CLUSTERROCKS."
echo
```

The following scripts shows the content of stop-hadoop-clusterrocks.sh:

```
[root@clusterrocks hadoop-2.6.0]# cat stop-hadoop-clusterrocks.sh
#! /bin/bash

# File: stop-hadoop-clusterrocks.sh
# Date: Thu Feb 20 23:08:52 MYT 2014
# Author: WRY

# OPENING MESSAGE
echo
echo "BISMILLAH. STOPPING HADOOP ON CLUSTERROCKS."
echo

# PROVIDE ACTUAL SITE SETTINGS
export HADOOP_CLUSTER_NAME=clusterrocks
export JAVA_HOME=/usr/java/latest
export HADOOP_PREFIX=/usr/local/hadoop-2.6.0
export HADOOP_CONF_DIR=/usr/local/hadoop-2.6.0/etc/hadoop
export HADOOP_YARN_HOME=/usr/local/hadoop-2.6.0

# DISPLAY ENVIRONMENT VARIABLES
echo "JAVA_HOME        = " $JAVA_HOME
echo "HADOOP_PREFIX    = " $HADOOP_PREFIX
echo "HADOOP_CONF_DIR  = " $HADOOP_CONF_DIR
echo "HADOOP_YARN_HOME = " $HADOOP_YARN_HOME
echo

echo "STOPPING FIVE(5) HADOOP SERVICES"
echo "================================"

echo
echo "Executing 1/5 ... ON MASTER"
$HADOOP_PREFIX/sbin/hadoop-daemon.sh --config $HADOOP_CONF_DIR --script
    hdfs stop namenode

echo
echo "Executing 2/5 ... ON MASTER AND SLAVES"
```

```
$HADOOP_PREFIX/sbin/hadoop-daemons.sh --config $HADOOP_CONF_DIR --script
    hdfs stop datanode

echo
echo "Executing 3/5 ... ON MASTER"
$HADOOP_YARN_HOME/sbin/yarn-daemon.sh --config $HADOOP_CONF_DIR stop
    resourcemanager

echo
echo "Executing 4/5 ... ON MASTER AND SLAVES"
$HADOOP_YARN_HOME/sbin/yarn-daemons.sh --config $HADOOP_CONF_DIR stop
    nodemanager

echo
echo "Executing 5/5 ... ON MASTER"
$HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh stop historyserver --config
    $HADOOP_CONF_DIR

# CLOSING MESSAGE
echo
echo "ALHAMDULILLAH. SUCCESS STOPPING HADOOP ON CLUSTERROCKS."
echo
```

6. The rest of the machines in the cluster act as both DataNode and NodeManager. These are the slaves.
   Once you have set your master you need to copy the Hadoop folder into the same directory of your slaves
   /usr/local

```
[taqi@clusterrocks hadoop]# sudo -i
[root@clusterrocks local]# cd /usr/local/
[root@clusterrocks local]# rsync -av ./hadoop-2.6.0 compute-0-0:/usr/local
[root@clusterrocks local]# rsync -av ./hadoop-2.6.0 compute-0-1:/usr/local
[root@clusterrocks local]# rsync -av ./hadoop-2.6.0 compute-0-2:/usr/local
...
REPEAT FOR ALL COMPUTE NODES
[root@clusterrocks local]# rsync -av ./hadoop-2.6.0 compute-0-3:/usr/local
[root@clusterrocks local]# rsync -av ./hadoop-2.6.0 compute-0-4:/usr/local
=====================================
After copying you need to updatedb
=====================================
[root@clusterrocks local]# updatedb
[root@clusterrocks local]# tentakel updatedb
```

7. Once we have completed configuration steps we need to format the Namenode for first time:

```
[taqi@clusterrocks hadoop]# sudo -i
[root@clusterrocks local]# cd /usr/local/hadoop-2.6.0
[root@clusterrocks hadoop-2.6.0]# bin/hdfs namenode -format
```

8. Now we are ready to run the Hadoop on Clusterrocks

```
1. Check if hadoop is already running on both master and slaves
=======================================
[root@clusterrocks hadoop-2.6.0]# jps
26472 Jps
[root@clusterrocks hadoop-2.6.0]# tentakel jps
### compute-0-2(stat: 0, dur(s): 0.63):
28351 Jps

### compute-0-3(stat: 0, dur(s): 0.63):
28391 Jps

....
....
....

### compute-0-1(stat: 0, dur(s): 0.66):
28392 Jps

### compute-0-9(stat: 0, dur(s): 10.13):
Remote command timed out on host compute-0-9

==========================================
2. Run start-hadoop-clusterrocks.sh script
====================================================
[root@clusterrocks hadoop-2.6.0]# ./start-hadoop-clusterrocks.sh

BISMILLAH. STARTING HADOOP ON CLUSTERROCKS.

Hadoop 2.6.0
Subversion http://svn.apache.org/repos/asf/hadoop/common -r 1616291
Compiled by jenkins on 2014-12-01T17:31Z
Compiled with protoc 2.6.0
From source with checksum 423dcd5a752eddd8e45ead6fd5ff9a24
This command was run using /usr/local/hadoop-2.6.0/share/hadoop/common/
    hadoop-common-2.6.0.jar

java version "1.7.0_03"
Java(TM) SE Runtime Environment (build 1.7.0_03-b04)
Java HotSpot(TM) 64-Bit Server VM (build 22.1-b02, mixed mode)
```

```
CLUSTER_NAME       =  clusterrocks
JAVA_HOME          =  /usr/java/latest
JSVC_HOME          =  /usr/java/latest
HADOOP_PREFIX      =  /usr/local/hadoop-2.5.0
HADOOP_CONF_DIR    =  /usr/local/hadoop-2.5.0/etc/hadoop
HADOOP_YARN_HOME   =  /usr/local/hadoop-2.6.0
# RUN ONCE /usr/local/hadoop-2.6.0/bin/hdfs namenode -format clusterrocks.
    mmu.edu.my

STARTING FIVE(5) REQUIRED HADOOP SERVICES
=========================================


Executing 1/5 ... RUN ON MASTER
starting namenode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-taqi-
    namenode-clusterrocks.mmu.edu.my.out

Executing 2/5 ... RUN ON MASTER AND SLAVES
clusterrocks: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-clusterrocks.mmu.edu.my.out
compute-0-3: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-3.local.out
compute-0-2: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-2.local.out
compute-0-0: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-0.local.out
compute-0-7: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-7.local.out
compute-0-5: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-5.local.out
compute-0-1: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-1.local.out
compute-0-8: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-8.local.out
compute-0-6: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-6.local.out
compute-0-9: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-9.local.out
compute-0-4: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/
    hadoop-taqi-datanode-compute-0-4.local.out

Executing 3/5 ... RUN ON MASTER
starting resourcemanager, logging to /usr/local/hadoop-2.6.0/logs/yarn-taqi
    -resourcemanager-clusterrocks.mmu.edu.my.out

Executing 4/5 ... RUN ON MASTER AND SLAVES
clusterrocks: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs
    /yarn-taqi-nodemanager-clusterrocks.mmu.edu.my.out
```

```
compute-0-5: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-5.local.out
compute-0-7: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-7.local.out
compute-0-6: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-6.local.out
compute-0-0: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-0.local.out
compute-0-3: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-3.local.out
compute-0-9: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-9.local.out
compute-0-8: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-8.local.out
compute-0-4: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-4.local.out
compute-0-2: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-2.local.out
compute-0-1: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/
    yarn-taqi-nodemanager-compute-0-1.local.out

Executing 5/5 ... RUN ON MASTER
chown: changing ownership of '/usr/local/hadoop-2.6.0/logs': Operation not
    permitted
starting historyserver, logging to /usr/local/hadoop-2.6.0/logs/mapred-taqi
    -historyserver-clusterrocks.mmu.edu.my.out


ALHAMDULILLAH. SUCCESS STARTING HADOOP ON CLUSTERROCKS.
===========================================================
3. Check Hadoop is running on both Master and Slave nodes
===========================================================


--> Check if Hadoop running on Master Node
===========================================================
[root@clusterrocks hadoop-2.6.0]# jps
701 Jps
32121 NameNode
32441 ResourceManager
32326 DataNode
32626 NodeManager
439 JobHistoryServer

--> Check if Hadoop running on All Slaves Node
===========================================================
--> Note : we only need to two service running on slaves which are DataNode
     and NodeManager
the rest is not necessary to be running on slave
```

```
[root@clusterrocks hadoop-2.6.0]#  tentakel jps
### compute-0-5(stat: 0, dur(s): 0.72):
1338 NodeManager
1473 Jps
1230 DataNode

### compute-0-3(stat: 0, dur(s): 0.72):
1375 NodeManager
1268 DataNode
1510 Jps
....
### compute-0-4(stat: 0, dur(s): 0.78):
1305 Jps
1058 DataNode
1166 NodeManager

### compute-0-8(stat: 0, dur(s): 0.81):
1375 NodeManager
1269 DataNode
1512 Jps




=================================
4. Once you are done with Hadoop stop Hadoop:
Because if you don't shut down the Hadoop other user can not run it.
=================================
[root@clusterrocks hadoop-2.6.0]#  /usr/local/hadoop-2.6.0/stop-hadoop-
    clusterrocks.sh

BISMILLAH. STOPPING HADOOP ON CLUSTERROCKS.

JAVA_HOME          =  /usr/java/latest
HADOOP_PREFIX      =  /usr/local/hadoop-2.6.0
HADOOP_CONF_DIR    =  /usr/local/hadoop-2.6.0/etc/hadoop
HADOOP_YARN_HOME   =  /usr/local/hadoop-2.6.0

STOPPING FIVE(5) HADOOP SERVICES
=================================


Executing 1/5 ... ON MASTER
stopping namenode

Executing 2/5 ... ON MASTER AND SLAVES
clusterrocks: stopping datanode
compute-0-1: stopping datanode
compute-0-3: stopping datanode
```

```
compute-0-8: stopping datanode
compute-0-5: stopping datanode
compute-0-2: stopping datanode
compute-0-6: stopping datanode
compute-0-7: stopping datanode
compute-0-0: stopping datanode
compute-0-9: stopping datanode
compute-0-4: stopping datanode

Executing 3/5 ... ON MASTER
stopping resourcemanager

Executing 4/5 ... ON MASTER AND SLAVES
clusterrocks: stopping nodemanager
compute-0-2: no nodemanager to stop
compute-0-1: no nodemanager to stop
compute-0-3: no nodemanager to stop
compute-0-6: no nodemanager to stop
compute-0-5: no nodemanager to stop
compute-0-8: no nodemanager to stop
compute-0-0: no nodemanager to stop
compute-0-7: no nodemanager to stop
compute-0-4: no nodemanager to stop
compute-0-9: no nodemanager to stop

Executing 5/5 ... ON MASTER
chown: changing ownership of '/usr/local/hadoop-2.6.0/logs': Operation not
    permitted
stopping historyserver

ALHAMDULILLAH. SUCCESS STOPPING HADOOP ON CLUSTERROCKS.
```

# Appendix C

# Word Count Application on Hadoop

In this appendix we are going to show how to run a word count application on Hadoop.
But before running the application we need to learn how to copy files into Hadoop File System (HDFS).

HDFS commands are mostly like Linux commands so since we are familiar with Linux it is not much difficult to learn the basic commands which are known as FS Shell on Hadoop website.

You can find a full list of these commands on the following URL: `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html`

1. The following scripts shows some basic commands of FS shell on Hadoop.

```
=====================================================
WORKING WITH HDFS (LISTING, COPYING AND READING FILE INTO HDFS)
=====================================================
taqi@taqi:~$ hdfs dfs -ls /
taqi@taqi:~$ hdfs dfs -mkdir /taqi/data/book
mkdir: '/taqi/data/book': No such file or directory
taqi@taqi:~$ hdfs dfs -mkdir -p /taqi/data/book
taqi@taqi:~$ hdfs dfs -mkdir -p /taqi/result/chapter1
taqi@taqi:~$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - taqi supergroup          0 2014-08-25 14:14 /taqi
taqi@taqi:~$ hdfs dfs -ls /taqi
Found 2 items
drwxr-xr-x   - taqi supergroup          0 2014-08-25 14:14 /taqi/data
drwxr-xr-x   - taqi supergroup          0 2014-08-25 14:14 /taqi/result
taqi@taqi:~$ hdfs dfs -ls /taqi/data
Found 1 items
drwxr-xr-x   - taqi supergroup          0 2014-08-25 14:14 /taqi/data/book
taqi@taqi:~$ hdfs dfs -ls /taqi/data/result
Found 1 items
drwxr-xr-x   - taqi supergroup          0 2014-08-25 14:14 /taqi/result/
    chapter1
taqi@taqi:~$ hdfs dfs -copyFromLocal /home/taqi/data/ThreeGables.txt /taqi/
    data/book
taqi@taqi:~$ hdfs dfs -ls /home/taqi/data/book
Found 1 items
-rw-r--r--   1 taqi supergroup  33344 2014-08-25 14:16 /taqi/data/book/
    ThreeGables.txt
taqi@taqi:~$ hdfs dfs -cat /taqi/data/book/ThreeGables.txt
:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:
          -----======Earth's Dreamlands======-----
             (313)558-5024 {14.4} (313)558-5517
                A BBS for text file junkies
                RPGNet GM File Archive Site
.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.


             The Adveniure of the Three Gables

  I don't think that any of my adventures with Mr. Sherlock
Holmes opened quite so abruptly, or so dramatically, as that
which I associate with The Three Gables. I had not seen Holmes
for some days and had no idea of the new channel into which his
activities had been directed. He was in a chatty mood that
morning, however, and had just settled me into the well-worn
low armchair on one side of the fire, while he had curled down
```

```
with his pipe in his mouth upon the opposite chair, when our
visitor arrived. If I had said that a mad bull had arrived it would
give a clearer impression of what occurred.
  ...................................
  "Very good. I think you will sign me a check for that, and I
will see that it comes to Mrs. Maberley. You owe her a little
change of air. Meantime, lady" -- he wagged a cautionary
forefinger -- "have a care! Have a care! You can't play with
edged tools forever without cutting those dainty hands."
```

2. In the following lines of codes we do the word count on our laptop (single machine) and in the next part we do the same word count on Clusterrocks

```
WHEN YOU PLACED YOUR FILE INTO YOUR HDFS YOU
CAN RUN A MAPREDUCE APPLICATION AGAINST YOUR
CLUSTER
--> IN OUR CASE IN THE BELOW SECTION WE ARE RUNNING
WORDCOUNT APPLICATION TO COUNT THE NUMBER OF OCCURANCE
 IN OUT NOVEL
=============================================

taqi@taqi:~$ hadoop jar /home/taqi/MapReduceFiles/bin/WordCount.jar
 /taqi/data/book /taqi/result/Chapter2
14/08/25 14:17:50 INFO client.RMProxy:
  Connecting to ResourceManager at /0.0.0.0:8032
14/08/25 14:17:50 INFO client.RMProxy:
  Connecting to ResourceManager at /0.0.0.0:8032
14/08/25 14:17:51 INFO mapred.FileInputFormat: Total input paths to process
     : 1
14/08/25 14:17:51 INFO mapreduce.JobSubmitter: number of splits:2
14/08/25 14:17:51 INFO mapreduce.JobSubmitter:
  Submitting tokens for job: job_1408947194469_0001
14/08/25 14:17:52 INFO impl.YarnClientImpl:
  Submitted application application_1408947194469_0001
14/08/25 14:17:52 INFO mapreduce.Job: The url to track the job:
 http://taqi:8088/proxy/application_1408947194469_0001/
14/08/25 14:17:52 INFO mapreduce.Job: Running job: job_1408947194469_0001
14/08/25 14:18:00 INFO mapreduce.Job:
  Job job_1408947194469_0001 running in uber mode : false
14/08/25 14:18:00 INFO mapreduce.Job:  map 0% reduce 0%
14/08/25 14:18:07 INFO mapreduce.Job:  map 100% reduce 0%
14/08/25 14:18:14 INFO mapreduce.Job:  map 100% reduce 100%
14/08/25 14:18:14 INFO mapreduce.Job:
  Job job_1408947194469_0001 completed successfully
14/08/25 14:18:14 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=23150
    FILE: Number of bytes written=325149
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=37350
    HDFS: Number of bytes written=13115
    HDFS: Number of read operations=9
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=2
    Launched reduce tasks=1
    Data-local map tasks=2
    Total time spent by all maps in occupied slots (ms)=10992
    Total time spent by all reduces in occupied slots (ms)=4045
    Total time spent by all map tasks (ms)=10992
    Total time spent by all reduce tasks (ms)=4045
    Total vcore-seconds taken by all map tasks=10992
    Total vcore-seconds taken by all reduce tasks=4045
    Total megabyte-seconds taken by all map tasks=11255808
    Total megabyte-seconds taken by all reduce tasks=4142080
  Map-Reduce Framework
    Map input records=642
    Map output records=6096
    Map output bytes=55065
    Map output materialized bytes=23156
    Input split bytes=198
    Combine input records=6096
    Combine output records=1821
    Reduce input groups=1451
    Reduce shuffle bytes=23156
    Reduce input records=1821
    Reduce output records=1451
    Spilled Records=3642
    Shuffled Maps =2
    Failed Shuffles=0
    Merged Map outputs=2
    GC time elapsed (ms)=196
    CPU time spent (ms)=4300
    Physical memory (bytes) snapshot=672321536
    Virtual memory (bytes) snapshot=2529718272
    Total committed heap usage (bytes)=560988160
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
```

```
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=37152
    File Output Format Counters
        Bytes Written=13115
taqi@taqi:~$ hdfs dfs -ls /taqi/result/data/Chapter2
Found 2 items
-rw-r--r--   1 taqi supergroup      0 2014-08-25 14:18 /taqi/result/Chapter2
    /_SUCCESS
-rw-r--r--   1 taqi supergroup  13115 2014-08-25 14:18 /taqi/result/
    Chapter2/part-00000
==================================================
ALTHOUGH FS SHELL HAS MANY COMMON WITH LINUX COMMANDS
IT DOES NOT MEAN ALL COMMANDS IN LINUX CAN BE EXECUTED
IN FS SHELL
EXAMPLE OF WRONG COMMAND
==================================================
taqi@taqi:~$ hdfs dfs -less /taqi/result/Chapter2/part-00000
-less: Unknown command
==================================================
READING OUR SAMPLE OUT PUT
==================================================
taqi@taqi:~$ hdfs dfs -cat/taqi/result/Chapter2/part-00000
a 145
about 11
above 1
abruptly  1
Ahmad 2
accounts  1
acre  1
across  1
act 2
actions 1
activities  1
............
you 143
youll 2
young 4
your  30
youre 1
yours 3
yourself  2
youve 1

taqi@taqi:~$ jps
8218 Jps
6356 ResourceManager
6200 SecondaryNameNode
```

```
6488 NodeManager
5770 NameNode
5922 DataNode
==============================================
ONCE YOU ARE DONE WITH HADOOP MAKE SURE YOU STOP
ALL SERVICES
==============================================
taqi@taqi:~$ stop-dfs.sh
taqi: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes
0.0.0.0: stopping secondarynamenode

taqi@taqi:~$ stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
localhost: nodemanager did not stop gracefully after 5 seconds: killing
    with kill -9
no proxyserver to stop
=============================================
ONCE YOU STOPPED THE SERVICES MAKE SURE THEY ALL HAVE BEEN KILLED
=============================================
taqi@taqi:~$ jps
8978 Jps

taqi@taqi:~$ "exit"
exit

Script done on Mon 25 Aug 2014 02:21:32 PM MYT
```

3. The following shows how to run the same word count that we did in previous section on Clusterrocks

```
=====================================================
Step One: Once all services are running perfectly try to create directories
     and store some files into HDFS
=====================================================
[taqi@clusterrocks hadoop-2.5.0]$ bin/hdfs dfs -mkdir /taqi/data/book

[taqi@clusterrocks ~]$ cd Download/
[taqi@clusterrocks Download]$ ll
total 136968
-rw-rw-r-- 1 taqi taqi 138656756 Aug  4 19:29 hadoop-2.4.1.tar.gz
-rw-rw-r-- 1 taqi taqi   1573078 Aug 22 04:43 JamesJoycesUlysses.txt
    <------------ We will copy this file inot HDfS
```

```
-rw-rw-r-- 1 taqi taqi      1516 Aug 22 04:44 WordCount.class
-rw-rw-r-- 1 taqi taqi      5808 Aug 22 04:44 WordCount.jar
-rw-rw-r-- 1 taqi taqi      2061 Aug 22 04:44 WordCount$Map.class
-rw-rw-r-- 1 taqi taqi      1591 Aug 22 04:44 WordCount$Reduce.class

[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -
    copyFromLocal ./JamesJoycesUlysses.txt /taqi/data/book/
===============================================
Step Two: Check if the file is copied to the required directory
===============================================
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -ls /
    taqi/data/book
Found 1 items
-rw-r--r--   3 taqi supergroup    1573078 2014-08-22 23:02 /taqi/data/book/
    JamesJoycesUlysses.txt


===============================================
Step Three: Make a directory for your result of you wordcount mapreduce
    application
===============================================
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -mkdir /
    taqi/data/result


===============================================
Step Four: Run the following WordCount.jar application against our cluster
The following is a MapReduce Job
===============================================
[taqi@clusterrocks Download]$ ll
total 136968
-rw-rw-r-- 1 taqi taqi 138656756 Aug  4 19:29 hadoop-2.4.1.tar.gz
-rw-rw-r-- 1 taqi taqi   1573078 Aug 22 04:43 JamesJoycesUlysses.txt
-rw-rw-r-- 1 taqi taqi      1516 Aug 22 04:44 WordCount.class
-rw-rw-r-- 1 taqi taqi      5808 Aug 22 04:44 WordCount.jar <-- This is our
     MapReduce Application that needs two argument
-rw-rw-r-- 1 taqi taqi      2061 Aug 22 04:44 WordCount$Map.class
-rw-rw-r-- 1 taqi taqi      1591 Aug 22 04:44 WordCount$Reduce.class


-----------------------------------------------
The Above application needs two arguement one is the directory of our input
     text file and another is the directory that we want our result

[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hadoop jar
    WordCount.jar /taqi/data/book /taqi/data/result/novel
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hadoop jar
    WordCount.jar /taqi/data/book /taqi/data/result/novel14/08/22 23:50:36
    WARN util.NativeCodeLoader: Unable to load native-hadoop library for
    your platform... using builtin-java classes where applicable
```

```
14/08/22 23:50:37 INFO Configuration.deprecation: session.id is deprecated.
     Instead, use dfs.metrics.session-id
14/08/22 23:50:37 INFO jvm.JvmMetrics: Initializing JVM Metrics with
    processName=JobTracker, sessionId=
14/08/22 23:50:38 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with
    processName=JobTracker, sessionId= - already initialized
14/08/22 23:50:38 WARN mapreduce.JobSubmitter: Hadoop command-line option
    parsing not performed. Implement the Tool interface and execute your
    application with ToolRunner to remedy this.
14/08/22 23:50:38 INFO mapred.FileInputFormat: Total input paths to process
     : 1
14/08/22 23:50:39 INFO mapreduce.JobSubmitter: number of splits:1
14/08/22 23:50:39 INFO mapreduce.JobSubmitter: Submitting tokens for job:
    job_local1655922483_0001
14/08/22 23:50:39 WARN conf.Configuration: file:/tmp/hadoop-taqi/mapred/
    staging/taqi1655922483/.staging/job_local1655922483_0001/job.xml:an
    attempt to override final parameter: mapreduce.job.end-notification.max
    .retry.interval; Ignoring.
...
14/08/22 23:50:41 INFO reduce.MergeManagerImpl: closeInMemoryFile -> map-
    output of size: 431825, inMemoryMapOutputs.size() -> 1, commitMemory ->
     0, usedMemory ->431825
14/08/22 23:50:41 INFO reduce.EventFetcher: EventFetcher is interrupted..
    Returning
14/08/22 23:50:41 INFO mapred.LocalJobRunner: 1 / 1 copied.
14/08/22 23:50:41 INFO reduce.MergeManagerImpl: finalMerge called with 1 in
    -memory map-outputs and 0 on-disk map-outputs
14/08/22 23:50:41 INFO mapred.Merger: Merging 1 sorted segments
14/08/22 23:50:41 INFO mapred.Merger: Down to the last merge-pass, with 1
    segments left of total size: 431821 bytes
14/08/22 23:50:41 INFO reduce.MergeManagerImpl: Merged 1 segments, 431825
    bytes to disk to satisfy reduce memory limit
14/08/22 23:50:41 INFO reduce.MergeManagerImpl: Merging 1 files, 431829
    bytes from disk
14/08/22 23:50:41 INFO reduce.MergeManagerImpl: Merging 0 segments, 0 bytes
     from memory into reduce
14/08/22 23:50:41 INFO mapred.Merger: Merging 1 sorted segments
14/08/22 23:50:41 INFO mapred.Merger: Down to the last merge-pass, with 1
    segments left of total size: 431821 bytes
14/08/22 23:50:41 INFO mapred.LocalJobRunner: 1 / 1 copied.
14/08/22 23:50:42 INFO mapred.Task: Task:
    attempt_local1655922483_0001_r_000000_0 is done. And is in the process
    of committing
14/08/22 23:50:42 INFO mapred.LocalJobRunner: 1 / 1 copied.
14/08/22 23:50:42 INFO mapred.Task: Task
    attempt_local1655922483_0001_r_000000_0 is allowed to commit now
...
14/08/22 23:50:42 INFO mapreduce.Job:  map 100% reduce 100%
```

```
14/08/22 23:50:42 INFO mapreduce.Job: Job job_local1655922483_0001
    completed successfully
14/08/22 23:50:42 INFO mapreduce.Job: Counters: 38
...
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=1573078
  File Output Format Counters
    Bytes Written=316024
==============================================
Step Five: upon successful run of application we will have the following
    result
==============================================
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -ls /
    taqi/data/result/novel/
Found 2 items
-rw-r--r--   3 taqi supergroup          0 2014-08-22 23:50 /taqi/data/
    result/novel/_SUCCESS
-rw-r--r--   3 taqi supergroup     316024 2014-08-22 23:50 /taqi/data/
    result/novel/part-00000
==============================================
Step Six: Reading the content of our result
==============================================
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -cat /
    taqi/data/result/novel/part-00000
Once you enter you will have the following result

wilt  2
wily  1
wimbles 1
wimple  2
win 13
wince 1
winced  1
winces  3
...
...........
wry 1  <----------Forced data
wrynecked 1
wud 1
wull  1
wus 1
wusser  1
wwwgutenbergorg 3
wy  2
```

```
wylie 7
wylies 1
wyndham 1
wynns 1
zut 1



==========================================
Running Word Count example against my chapter 2 of FYp
==========================================
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hdfs dfs -rmr /
    taqi/data/result/Chapter2
rmr: DEPRECATED: Please use 'rm -r' instead.
14/08/23 00:28:19 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where
    applicable
14/08/23 00:28:20 INFO fs.TrashPolicyDefault: Namenode trash configuration:
     Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /taqi/data/result/Chapter2
[taqi@clusterrocks Download]$ /usr/local/hadoop-2.5.0/bin/hadoop jar ./
    WordCount.jar /taqi/data/book /taqi/data/result/Chapter2
14/08/23 00:28:38 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where
    applicable
14/08/23 00:28:38 INFO Configuration.deprecation: session.id is deprecated.
     Instead, use dfs.metrics.session-id
14/08/23 00:28:38 INFO jvm.JvmMetrics: Initializing JVM Metrics with
    processName=JobTracker, sessionId=
14/08/23 00:28:40 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with
    processName=JobTracker, sessionId= - already initialized
14/08/23 00:28:40 WARN mapreduce.JobSubmitter: Hadoop command-line option
    parsing not performed. Implement the Tool interface and execute your
    application with ToolRunner to remedy this
....
14/08/23 00:28:45 INFO mapreduce.Job:  map 100% reduce 100%
14/08/23 00:28:45 INFO mapreduce.Job: Job job_local2002818904_0001
    completed successfully
14/08/23 00:28:45 INFO mapreduce.Job: Counters: 38
  File System Counters
    FILE: Number of bytes read=34964
    FILE: Number of bytes written=506621
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=34330
    HDFS: Number of bytes written=8232
    HDFS: Number of read operations=15
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
  Map-Reduce Framework
    Map input records=158
    Map output records=2745
    Map output bytes=27472
    Map output materialized bytes=11437
    Input split bytes=115
    Combine input records=2745
    Combine output records=810
    Reduce input groups=810
    Reduce shuffle bytes=11437
    Reduce input records=810
    Reduce output records=810
    Spilled Records=1620
    Shuffled Maps =1
    Failed Shuffles=0
    Merged Map outputs=1
    GC time elapsed (ms)=1503
    CPU time spent (ms)=0
    Physical memory (bytes) snapshot=0
    Virtual memory (bytes) snapshot=0
    Total committed heap usage (bytes)=476971008
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=17165
  File Output Format Counters
    Bytes Written=8232
==================================================
Sample output
====================================================
handle  1
handling  1
hard  1
hardwares 1
has 6
have  7
having  1
hdfs  17
heartbeats  2
.....
ruslan  1 <----------- One time occurance of Ruslan
same  4
```

```
save  1
saved 1

....
takes 1
talk  3
taqi  4  <———— Four times occurance of word taqi
target  1
task  4
tasks 2
...
writing 1
wry 1 <—— One occurance of WRY name :D
xml 1
yahoo 1
yarn  1
yarnbased 1
you 15
your  3
zettabytescitebigdch  1
"readable"  1
"volume"  1
```

# Appendix D

# Spark Single Node Setup

In this appendix we explain how to install and configure the Spark on the Single Node Machine.

1. Download the spark-VERSION-bin-hadoopVERSION.tar.gz from the following URL:
   `http://spark.apache.org/downloads.html`
   where VERSION is the number of Spark and Hadoop you want.

2. Extract it and copy into /usr/local

```
taqi@taqi-desktop:~/Downloads$ ls -l |grep spark
-rw-r--r-- 1 taqi taqi  191607044 Nov  29 17:46 spark-1.1.1-bin-hadoop2.4.
    tgz
taqi@taqi-desktop:~/Downloads$ tar xvf spark-1.1.1-bin-hadoop2.4.tgz
spark-1.1.1-bin-hadoop2.4/
...
taqi@taqi-desktop:~/Downloads$ ls -l |grep spark
drwxrwxr-x 9 taqi taqi       4096 Nov  20 05:08 spark-1.1.1-bin-hadoop2.4
-rw-r--r-- 1 taqi taqi  191607044 Nov  29 17:46 spark-1.1.1-bin-hadoop2.4.
    tgz
taqi@taqi-desktop:~/Downloads$ sudo mv spark-1.1.1-bin-hadoop2.4 /usr/local
    /
taqi@taqi-desktop:~/Downloads$ cd /usr/local/
taqi@taqi-desktop:/usr/local$ ls -l |grep spark
drwx------ 12 taqi taqi 4096 Jan  10 05:47 spark-1.1.1-bin-hadoop2.4
```

3. Set environmental variables

```
taqi@taqi-desktop:~$ nano .bashrc
Paste the following in the opened files
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize
    =512m"
export SPARK=/usr/local/spark-1.1.0-bin-hadoop2.4/
export PATH=$PATH:$SPARK/bin
=================================
save the file and close it by
Ctrl + X and yes
================================
```

4. To run Spark command prompt run the following:

```
taqi@taqi-desktop:/usr/local/spark-1.1.1-bin-hadoop2.4$ bin/spark-shell
/usr/local/spark-1.1.1-bin-hadoop2.4/conf/spark-env.sh: line 22:
    HADOOP_CONF_DIR: command not found
/usr/local/spark-1.1.1-bin-hadoop2.4/conf/spark-env.sh: line 23: /usr/local
    /hadoop-2.4.1/etc/hadoop/: Is a directory
Spark assembly has been built with Hive, including Datanucleus jars on
    classpath
Welcome to
      ----              --
     / --/--   ---  -----/ /--
    -\ \/ - \/ - ‘/ --/   ’-/
   /---/ .--/\-,-/-/ /-/\-\   version 1.1.1
      /-/

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_65)
Type in expressions to have them evaluated.
Type :help for more information.
15/01/28 06:55:46 WARN Utils: Your hostname, taqi-desktop resolves to a
    loopback address: 127.0.1.1; using 192.168.0.108 instead (on interface
    wlan0)
15/01/28 06:55:46 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to
    another address
15/01/28 06:55:46 WARN NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where
    applicable
Spark context available as sc.

scala>
```

5. In order to submit application we need to run the start-all.sh script on sbin directory of Spark

```
taqi@taqi-desktop:/usr/local/spark-1.1.1-bin-hadoop2.4$ sbin/start-all.sh
/usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh: line 22:
    HADOOP_CONF_DIR: command not found
/usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh: line 23: /
    usr/local/hadoop-2.4.1/etc/hadoop/: Is a directory
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark
    -1.1.1-bin-hadoop2.4/sbin/../logs/spark-taqi-org.apache.spark.deploy.
    master.Master-1-taqi-desktop.out
/usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh: line 22:
    HADOOP_CONF_DIR: command not found
/usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh: line 23: /
    usr/local/hadoop-2.4.1/etc/hadoop/: Is a directory
localhost: /usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh:
    line 22: HADOOP_CONF_DIR: command not found
localhost: /usr/local/spark-1.1.1-bin-hadoop2.4/sbin/../conf/spark-env.sh:
    line 23: /usr/local/hadoop-2.4.1/etc/hadoop/: Is a directory
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/
    local/spark-1.1.1-bin-hadoop2.4/sbin/../logs/spark-taqi-org.apache.
    spark.deploy.worker.Worker-1-taqi-desktop.out
taqi@taqi-desktop:/usr/local/spark-1.1.1-bin-hadoop2.4$ jps
9134 Worker
8803 Master
9234 Jps
taqi@taqi-desktop:/usr/local/spark-1.1.1-bin-hadoop2.4$
```

6. While Spark is Running go the following URL to see the workers and status of applications that are running under Spark after submitting them to Spark:
   `http://localhost:8080/`

7. Submitting Application to spark is shown on the following scripts:

```
Refer to the next appendix
```

8. After submitting the application and while running the application on Spark we are able to see the status of individual applications and in which stage the application is on the following URL:
   `https://localhost:4040/`

# Appendix E

# Spark Cluster Setup

In this Appendix we have put the scripts for installing Hadoop on Clusterrocks

1. The following is Spark-1.0.2 installation on clusterrocks

```
wruslan@wruslan-fujitsu-ub1204:~$ ssh root@clusterrocks.mmu.edu.my
Password:
Last login: Sat Aug 23 03:53:56 2014 from clusterrocks.mmu.edu.my
Rocks 6.1 (Emerald Boa)
Profile built 10:52 14-Feb-2014

Kickstarted 19:47 14-Feb-2014
[root@clusterrocks ~]# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

[root@clusterrocks ~]# date
Sat Aug 23 18:10:45 MYT 2014
[root@clusterrocks ~]# uname -a
Linux clusterrocks.mmu.edu.my 2.6.32-279.14.1.el6.x86_64 #1 SMP Tue Nov 6
    23:43:09 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
[root@clusterrocks ~]#

=====================================================================

[root@clusterrocks local]# wget -c http://d3kbcqa49mib13.cloudfront.net/
    spark-1.0.2-bin-hadoop2.tgz
```

```
--2014-08-23 18:14:04--  http://d3kbcqa49mib13.cloudfront.net/spark-1.0.2-
    bin-hadoop2.tgz
Resolving d3kbcqa49mib13.cloudfront.net... 54.230.150.29, 54.230.150.32,
    54.230.150.50, ...
Connecting to d3kbcqa49mib13.cloudfront.net|54.230.150.29|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 187476648 (179M) [application/x-compressed]
Saving to: "spark-1.0.2-bin-hadoop2.tgz"
100%[=====================================================================
     187,476,648 6.48M/s   in 48s
2014-08-23 18:14:52 (3.70 MB/s) - "spark-1.0.2-bin-hadoop2.tgz" saved
    [187476648/187476648]

==========================================================================
[root@clusterrocks local]# ls -al
total 183240
drwxr-xr-x 26 root  root       4096 Aug 23 18:14 .
drwxr-xr-x 17 root  root       4096 May 14 20:31 ..
drwxr-xr-x  2 root  root       4096 Apr 10 21:27 bin
drwxr-xr-x  9 root  root       4096 Feb 26 17:53 cassandra-2.0.5
drwxr-xr-x  2 root  root       4096 Sep 23  2011 etc
drwxr-xr-x  2 root  root       4096 Sep 23  2011 games
drwxr-xr-x 10 root  root       4096 Feb 26 13:46 gnat2013
drwxr-xr-x  7 root  root       4096 Mar  7 18:03 gprolog-1.4.4
drwxrwxrwx 11 root  root       4096 Feb 24 18:55 hadoop-2.2.0
drwxrwxrwx 10 root  root       4096 Aug 20 04:12 hadoop-2.4.1
drwxrwxrwx 10 root  root       4096 Aug 22 20:52 hadoop-2.5.0
drwxr-xr-x  6 27429  603       4096 Nov  6  2013 hdf5-1.8.12
drwxr-xr-x  6 root  root       4096 Apr 10 21:27 include
drwxr-xr-x  7 root  root       4096 Apr 10 21:27 lib
drwxr-xr-x  2 root  root      57344 Mar  8 13:58 lib64
drwxr-xr-x  3 root  root       4096 Mar  8 18:00 libexec
drwxr-xr-x  8 root  root       4096 Mar  7 02:55 libsvm-3.17
drwxr-xr-x  3 root  root       4096 Apr 10 21:27 man
drwxr-xr-x 14  1000 1000       4096 Mar  8 18:08 octave-3.8.1
drwxr-xr-x  8 root  root       4096 Mar  6 11:07 pybrain0.3.1
drwxr-xr-x  6 root  root       4096 Mar  7 04:05 run-mdp-bimdp-oger-pybrain
drwxr-xr-x  2 root  root       4096 Sep 23  2011 sbin
drwxr-xr-x  7  1000 1000       4096 Mar 30  2013 scilab-5.4.1
drwxr-xr-x  7  1000 1000       4096 Mar 30  2013 scilab-5.4.1-32bit
drwxr-xr-x 13 root  root       4096 Mar  8 17:59 share
-rw-r--r--  1 root  root  187476648 Aug  6 04:38 spark-1.0.2-bin-hadoop2.tgz
drwxr-xr-x  2 root  root       4096 Sep 23  2011 src
[root@clusterrocks local]#
==========================================================================

[root@clusterrocks local]# gunzip spark-1.0.2-bin-hadoop2.tgz
[root@clusterrocks local]# tar -xf spark-1.0.2-bin-hadoop2.tar
```

```
[root@clusterrocks local]# updatedb
[root@clusterrocks local]# cd spark-1.0.2-bin-hadoop2/
[root@clusterrocks spark-1.0.2-bin-hadoop2]# ls -al
total 424
drwxrwxr-x  9 1000 1000   4096 Jul 26 06:30 .
drwxr-xr-x 27 root root   4096 Aug 23 18:17 ..
drwxrwxr-x  2 1000 1000   4096 Jul 26 06:30 bin
-rw-rw-r--  1 1000 1000 325351 Jul 26 06:30 CHANGES.txt
drwxrwxr-x  2 1000 1000   4096 Jul 26 06:30 conf
drwxrwxr-x  4 1000 1000   4096 Jul 26 06:30 ec2
drwxrwxr-x  3 1000 1000   4096 Jul 26 06:30 examples
drwxrwxr-x  2 1000 1000   4096 Jul 26 06:30 lib
-rw-rw-r--  1 1000 1000  29983 Jul 26 06:30 LICENSE
-rw-rw-r--  1 1000 1000  22559 Jul 26 06:30 NOTICE
drwxrwxr-x  6 1000 1000   4096 Jul 26 06:30 python
-rw-rw-r--  1 1000 1000   4221 Jul 26 06:30 README.md
-rw-rw-r--  1 1000 1000     35 Jul 26 06:30 RELEASE
drwxrwxr-x  2 1000 1000   4096 Jul 26 06:30 sbin
[root@clusterrocks spark-1.0.2-bin-hadoop2]#

===============================================================

[root@clusterrocks conf]# pwd
/usr/local/spark-1.0.2-bin-hadoop2/conf
[root@clusterrocks conf]# nano spark-env.sh

JAVA_HOME=/usr/java/latest
JSVC_HOME=/usr/java/latest
HADOOP_CLUSTER_NAME=clusterrocks
HADOOP_PREFIX=/usr/local/hadoop-2.5.0
HADOOP_CONF_DIR=/usr/local/hadoop-2.5.0/etc/hadoop
HADOOP_YARN_HOME=/usr/local/hadoop-2.5.0
HADOOP_HOME=/usr/local/hadoop-2.5.0
HADOOP_MAPRED_HOME=/usr/local/hadoop-2.5.0
HADOOP_COMMON_HOME=/usr/local/hadoop-2.5.0
HADOOP_HDFS_HOME=/usr/local/hadoop-2.5.0

SPARK_MASTER_IP=10.106.17.102
SPARK_HADOOP_VERSION=2.2.0 SPARK_YARN=true sbt/sbt assembly
SPARK_HOME=/usr/local/spark-1.0.2-bin-hadoop2
FWDIR=/usr/local/spark-1.0.2-bin-hadoop2
SPARK_PREFIX=/usr/local/spark-1.0.2-bin-hadoop2
SPARK_CONF_DIR=/usr/local/spark-1.0.2-bin-hadoop2
SPARK_CLASSPATH=/usr/local/spark-1.0.2-bin-hadoop2
SPARK_SUBMIT_CLASSPATH=/usr/local/spark-1.0.2-bin-hadoop2

[root@clusterrocks conf]# updatedb
[root@clusterrocks local]# chown -R root.root spark-1.0.2-bin-hadoop2
```

```
[root@clusterrocks local]# chmod -R 777 spark-1.0.2-bin-hadoop2
[root@clusterrocks local]# updatedb
[root@clusterrocks local]#

Need to RSYNC to all compute nodes

[root@clusterrocks local]# rsync -av spark-1.0.2-bin-hadoop2 compute-0-0:/
    usr/local
[root@clusterrocks local]# rsync -av spark-1.0.2-bin-hadoop2 compute-0-1:/
    usr/local
....
UP TO compute-0-9
```

2. The following shows how to run spark on Clusterrocks

```
[root@clusterrocks local]# cd spark-1.0.2-bin-hadoop2/sbin

[root@clusterrocks sbin]# ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark
    -1.0.2-bin-hadoop2/sbin/../logs/spark-root-org.apache.spark.deploy.
    master.Master-1-clusterrocks.mmu.edu.my.out
clusterrocks: starting org.apache.spark.deploy.worker.Worker, logging to /
    usr/local/spark-1.0.2-bin-hadoop2/logs/spark-root-org.apache.spark.
    deploy.worker.Worker-1-clusterrocks.mmu.edu.my.out
compute-0-6: starting org.apache.spark.deploy.worker.Worker, logging to /
    usr/local/spark-1.0.2-bin-hadoop2/logs/spark-root-org.apache.spark.
    deploy.worker.Worker-1-compute-0-6.local.out
...
...
compute-0-4: starting org.apache.spark.deploy.worker.Worker, logging to /
    usr/local/spark-1.0.2-bin-hadoop2/logs/spark-root-org.apache.spark.
    deploy.worker.Worker-1-compute-0-4.local.out

[root@clusterrocks sbin]# jps
14169 Master
14591 Jps
14393 Worker

[root@clusterrocks sbin]# tentakel jps
### compute-0-6(stat: 0, dur(s): 0.55):
12831 Worker
12908 Jps
.....
.....
### compute-0-8(stat: 0, dur(s): 0.64):
13089 Jps
13012 Worker
```

```
[root@clusterrocks sbin]#
================================
Once you are done with spark run stop-all.sh to stop the Workers

[root@clusterrocks sbin]# ./stop-all.sh
clusterrocks: stopping org.apache.spark.deploy.worker.Worker
compute-0-3: stopping org.apache.spark.deploy.worker.Worker
...
...
compute-0-7: stopping org.apache.spark.deploy.worker.Worker
stopping org.apache.spark.deploy.master.Master
[root@clusterrocks sbin]#
```

3. Once you have all Workers and Master running visit the following URL to see the status of each Worker and Master and also the applications' status after submitting them to Spark:
   `https://clusterrocks.mmu.edu.my:8080/`

4. Submitting Application to spark is shown on the following scripts:

```
$ YOUR_SPARK_HOME/bin/spark-submit \
  --class YOUR_MAIN_CLASS_NAME \
  --master local[4] \
  PATH/TO/YOUR/APPLICATION.jar
```

5. After submitting the application and while running the application on Spark we are able to see the status of individual applications and in which stage the application is on the following URL:
   `https://clusterrocks.mmu.edu.my:4040/`

# Appendix F

# Data Preparation for Classification

In this section we show how we prepared our dataset to run the classification on it.

```
Data preparation steps on Terminal
=============================================


taqi@taqi-laptop:~/report/Files/original$ ll
total 5872
drwxrwxr-x 2 taqi taqi    4096 Jan  15 12:32 ./
drwxrwxr-x 5 taqi taqi    4096 Jan  15 12:32 ../
-rw-rw-r-- 1 taqi taqi 3974304 Jan  15 11:51 adult.data
-rw-rw-r-- 1 taqi taqi    5229 Jan  15 11:51 adult.names
-rw-rw-r-- 1 taqi taqi 2003131 Jan  15 11:51 adult.test
-rw-rw-r-- 1 taqi taqi     924 Jan  15 12:32 CreateFile.java
-rw-rw-r-- 1 taqi taqi    5558 Jan  15 12:32 DataPreparation.java

==============================================
Compile javacode
==============================================
taqi@taqi-laptop:~/report/Files/original$ javac *.java
taqi@taqi-laptop:~/report/Files/original$ ll
total 5880
drwxrwxr-x 2 taqi taqi    4096 Jan  15 12:33 ./
drwxrwxr-x 5 taqi taqi    4096 Jan  15 12:32 ../
-rw-rw-r-- 1 taqi taqi 3974304 Jan  15 11:51 adult.data
-rw-rw-r-- 1 taqi taqi    5229 Jan  15 11:51 adult.names
```

```
-rw-rw-r-- 1 taqi taqi 2003131 Jan  15 11:51 adult.test
-rw-rw-r-- 1 taqi taqi    1115 Jan  15 12:33 CreateFile.class
-rw-rw-r-- 1 taqi taqi     924 Jan  15 12:32 CreateFile.java
-rw-rw-r-- 1 taqi taqi    2912 Jan  15 12:33 DataPreparation.class
-rw-rw-r-- 1 taqi taqi    5558 Jan  15 12:32 DataPreparation.java
==================================================
Combine two files together
==================================================
taqi@taqi-laptop:~/report/Files/original$ cat adult.data >> rawData.txt
taqi@taqi-laptop:~/report/Files/original$ cat adult.test >> rawData.txt
taqi@taqi-laptop:~/report/Files/original$ ll
total 11720
drwxrwxr-x 2 taqi taqi    4096 Jan  15 12:35 ./
drwxrwxr-x 5 taqi taqi    4096 Jan  15 12:32 ../
-rw-rw-r-- 1 taqi taqi 3974304 Jan  15 11:51 adult.data
-rw-rw-r-- 1 taqi taqi    5229 Jan  15 11:51 adult.names
-rw-rw-r-- 1 taqi taqi 2003131 Jan  15 11:51 adult.test
-rw-rw-r-- 1 taqi taqi    1115 Jan  15 12:33 CreateFile.class
-rw-rw-r-- 1 taqi taqi     924 Jan  15 12:32 CreateFile.java
-rw-rw-r-- 1 taqi taqi    2912 Jan  15 12:33 DataPreparation.class
-rw-rw-r-- 1 taqi taqi    5558 Jan  15 12:32 DataPreparation.java
-rw-rw-r-- 1 taqi taqi 5977435 Jan  15 12:35 rawData.txt
taqi@taqi-laptop:~/report/Files/original$ wc -l adult.data adult.test
  32561 adult.data
  16281 adult.test
  48842 total
taqi@taqi-laptop:~/report/Files/original$ wc -l rawData.txt
48842 rawData.txt
==================================================
Run java program choose option 1 and give the pathof rawData.txt to it
Screen shot in dataPreparationFolder
==================================================
taqi@taqi-laptop:~/report/Files/original$ pwd
/home/taqi/report/Files/original
taqi@taqi-laptop:~/report/Files/original$ ll
total 18168
drwxrwxr-x 2 taqi taqi    4096 Jan  15 12:41 ./
drwxrwxr-x 5 taqi taqi    4096 Jan  15 12:32 ../
-rw-rw-r-- 1 taqi taqi 3974304 Jan  15 11:51 adult.data
-rw-rw-r-- 1 taqi taqi    5229 Jan  15 11:51 adult.names
-rw-rw-r-- 1 taqi taqi 2003131 Jan  15 11:51 adult.test
-rw-rw-r-- 1 taqi taqi    1115 Jan  15 12:33 CreateFile.class
-rw-rw-r-- 1 taqi taqi     924 Jan  15 12:32 CreateFile.java
-rw-rw-r-- 1 taqi taqi    2912 Jan  15 12:33 DataPreparation.class
-rw-rw-r-- 1 taqi taqi    5558 Jan  15 12:32 DataPreparation.java
-rw-rw-r-- 1 taqi taqi 6601275 Jan  15 12:41 rawDataNumberedFile.txt
-rw-rw-r-- 1 taqi taqi 5977435 Jan  15 12:35 rawData.txt
taqi@taqi-laptop:~/report/Files/original$
```

```
==========================================================
Count the number of labels
Counter and Screen shot is in dataPrepartion Folder
==========================================================


Split rawData into two set of files train.txt and test.txt
Each contain 50% of rawData
48842/2 = 24421
============================================================
taqi@taqi-laptop:~/report/Files/original$ split -l 24421 rawDataNumberedFile.txt
taqi@taqi-laptop:~/report/Files/original$ ll
total 24620
drwxrwxr-x 2 taqi taqi    4096 Jan  15 12:53 ./
drwxrwxr-x 5 taqi taqi    4096 Jan  15 12:32 ../
-rw-rw-r-- 1 taqi taqi 3974304 Jan  15 11:51 adult.data
-rw-rw-r-- 1 taqi taqi    5229 Jan  15 11:51 adult.names
-rw-rw-r-- 1 taqi taqi 2003131 Jan  15 11:51 adult.test
-rw-rw-r-- 1 taqi taqi    1115 Jan  15 12:33 CreateFile.class
-rw-rw-r-- 1 taqi taqi     924 Jan  15 12:32 CreateFile.java
-rw-rw-r-- 1 taqi taqi    2912 Jan  15 12:33 DataPreparation.class
-rw-rw-r-- 1 taqi taqi    5558 Jan  15 12:32 DataPreparation.java
-rw-rw-r-- 1 taqi taqi 6601275 Jan  15 12:41 rawDataNumberedFile.txt
-rw-rw-r-- 1 taqi taqi 5977435 Jan  15 12:35 rawData.txt
-rw-rw-r-- 1 taqi taqi 3287298 Jan  15 12:53 xaa
-rw-rw-r-- 1 taqi taqi 3313977 Jan  15 12:53 xab
taqi@taqi-laptop:~/report/Files/original$ wc -l xaa xab
  24421 xaa
  24421 xab
  48842 total
taqi@taqi-laptop:~/report/Files/original$ wc -l rawDataNumberedFile.txt
48842 rawDataNumberedFile.txt
taqi@taqi-laptop:~/report/Files/original$ mv xaa train.txt
taqi@taqi-laptop:~/report/Files/original$ mv xab test.txt
taqi@taqi-laptop:~/report/Files/original$ wc -l test.txt train.txt
  24421 test.txt
  24421 train.txt
  48842 total
==============================================================
Count the label in both files using java program
Screen shot is saved in dataPreparation folder
------------------------
Count result of train.txt from NetBeans
/home/taqi/report/Files/original/train.txt
Lines with >50K Label: 5853
Lines with <=50K Label: 18568
Lines with Unknown Label: 0
Total: 24421
```

```
Count result of test.txt from NetBeans
/home/taqi/report/Files/original/test.txt
Lines with >50K Label: 5834
Lines with <=50K Label: 18587
Lines with Unknown Label: 0
Total: 24421
================================================
Split the train into two separate files based on the label
using java program
Screenshot is available in dataPreparation folder
------------------------------------------------
Output of NetBeans for >50K
Enter path to file
/home/taqi/report/Files/original/train.txt
Files have been splitted successfully for >50K
Number of records in new file : 5853
--------------------------------
Output of NetBeans for >50K
Enter path to file
/home/taqi/report/Files/original/train.txt
Files have been splitted successfully for <=50K
Number of records in new file : 18568
BUILD SUCCESSFUL (total time: 19 seconds)

=====================================
Back in shell count the number of lines of two separated files
=====================================
taqi@taqi-laptop:~/report/Files/original$ wc -l trainLessThan50K.txt
    trainMoreThan50K.txt
  18568 trainLessThan50K.txt
   5853 trainMoreThan50K.txt
  24421 total
```

# Appendix G

# Classification on Spark

This appendix shows how we ran classification algorithms against our dataset that we prepared in appendix F

```
1. Run Hadoop-2.5.0
==============================

[root@clusterrocks hadoop-2.5.0]# jps
18301 NodeManager
18146 ResourceManager
2205 DTGateway
18041 DataNode
17866 NameNode
18559 JobHistoryServer
18696 Jps
[root@clusterrocks hadoop-2.5.0]# tentakel jps
### compute-0-0(stat: 0, dur(s): 0.63):
612 Jps
370 DataNode
467 NodeManager
...
...
### compute-0-9(stat: 0, dur(s): 0.74):
21027 DataNode
21120 NodeManager
21262 Jps


=================================================
2. Make directory for train dataset on clusterrocks
=================================================

[root@clusterrocks hadoop-2.5.0]# bin/hdfs dfs -ls /
```

```
15/01/19 19:13:28 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where applicable
Found 1 items
drwxrwx---   - root supergroup          0 2015-01-19 19:12 /tmp
[root@clusterrocks hadoop-2.5.0]# bin/hdfs dfs -mkdir /train
15/01/19 19:13:43 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where applicable
===============================================
3. Copy train dataset into HDFS
===============================================
[root@clusterrocks hadoop-2.5.0]# bin/hdfs dfs -copyFromLocal /home/taqi/Download
    /train* /train/
15/01/19 19:17:49 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where applicable
===============================================
4. Check if you have successfull copied and you have them in
HDFS by listing the content in /train directory
===============================================
[root@clusterrocks hadoop-2.5.0]# bin/hdfs dfs -ls /train
15/01/19 19:18:06 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   3 root supergroup    2124329 2015-01-19 19:17 /train/
    trainLessThan50K.txt
-rw-r--r--   3 root supergroup     691508 2015-01-19 19:17 /train/
    trainMoreThan50K.txt
===============================================
5. Run Spark Master
===============================================
[root@clusterrocks hadoop-2.5.0]# cd ..
[root@clusterrocks local]# cd spark-1.1.1-bin-hadoop2.4/

[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark
    -1.1.1-bin-hadoop2.4/sbin/../logs/spark-root-org.apache.spark.deploy.master.
    Master-1-clusterrocks.mmu.edu.my.out
==============================================
6. Check if Master is running
==============================================
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# jps
18301 NodeManager
20438 Jps
20335 Master
18146 ResourceManager
2205 DTGateway
18041 DataNode
17866 NameNode
18559 JobHistoryServer
```

```
==========================================
7. Start Spark workers on 10 nodes + master itself as worker =11 workers
==========================================
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# sbin/start-slaves.sh
clusterrocks: starting org.apache.spark.deploy.worker.Worker, logging to /usr/
    local/spark-1.1.1-bin-hadoop2.4/logs/spark-root-org.apache.spark.deploy.
    worker.Worker-1-clusterrocks.mmu.edu.my.out
compute-0-0: starting org.apache.spark.deploy.worker.Worker, logging to /usr/
    local/spark-1.1.1-bin-hadoop2.4/logs/spark-root-org.apache.spark.deploy.
    worker.Worker-1-compute-0-0.local.out
.......
compute-0-9: starting org.apache.spark.deploy.worker.Worker, logging to /usr/
    local/spark-1.1.1-bin-hadoop2.4/logs/spark-root-org.apache.spark.deploy.
    worker.Worker-1-compute-0-9.local.out
==========================================
8. Check if workers are working
==========================================
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# jps
18301 NodeManager
20595 Worker
20699 Jps
20335 Master
18146 ResourceManager
2205 DTGateway
18041 DataNode
17866 NameNode
18559 JobHistoryServer
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# tentakel jps
### compute-0-2(stat: 0, dur(s): 0.71):
18385 Worker
18463 Jps
18016 DataNode
18109 NodeManager

.....
.....

### compute-0-9(stat: 0, dur(s): 0.78):
21027 DataNode
21120 NodeManager
21398 Worker
21475 Jps
==========================================
9. Copy spark application jar file into clusterrocks
==========================================

==========================================
10. Submit the application to spark
```

```
========================================
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# pwd
/usr/local/spark-1.1.1-bin-hadoop2.4

[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# ./bin/spark-submit --class org.fyp
    .sparkbasics.LogisticRegression --master spark://10.106.17.102:7077 /home/
    taqi/Download/FinalYearPorject/target/FinalYearPorject-1.0-SNAPSHOT.jar

15/01/19 21:33:31 INFO storage.BlockManager: Removing block broadcast_202
15/01/19 21:33:31 INFO storage.MemoryStore: Block broadcast_202 of size 86696
    dropped from memory (free 277471237)
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    clusterrocks.mmu.edu.my:43810 in memory (size: 7.4 KB, free: 263.6 MB)
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    compute-0-3.local:39368 in memory (size: 7.4 KB, free: 265.1 MB)
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    compute-0-9.local:33492 in memory (size: 7.4 KB, free: 265.1 MB)
15/01/19 21:33:31 INFO spark.ContextCleaner: Cleaned broadcast 202
Out of 24421 Predictions:
  20183 Correct!
   4238 Wrong!

Out of 5834 Salaries >50k : 2449 Correctly Predicted

Out of 5834 Salaries >50k : 3385 Wrongly Predicted

Out of 18587 Salaries  <=50k : 17734 Correctly Predicted

Out of 18587 Salaries  <=50k : 853 Wrongly Predicted


========================================
11. Look at in WebUI to see if the appllication can run or not
========================================


12. Submit application under constraints of 10G executor-memory and 8 cores per
    node
========================================
[root@clusterrocks spark-1.1.1-bin-hadoop2.4]# ./bin/spark-submit --class org.fyp
    .sparkbasics.LogisticRegression --master spark://10.106.17.102:7077 --
    executor-memory 10G --total-executor-cores 8 /home/taqi/Download/
    FinalYearPorject/target/FinalYearPorject-1.0-SNAPSHOT.jar
.........
15/01/19 21:33:31 INFO storage.BlockManager: Removing block broadcast_202
15/01/19 21:33:31 INFO storage.MemoryStore: Block broadcast_202 of size 86696
    dropped from memory (free 277471237)
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    clusterrocks.mmu.edu.my:43810 in memory (size: 7.4 KB, free: 263.6 MB)
```

```
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    compute-0-3.local:39368 in memory (size: 7.4 KB, free: 265.1 MB)
15/01/19 21:33:31 INFO storage.BlockManagerInfo: Removed broadcast_202_piece0 on
    compute-0-9.local:33492 in memory (size: 7.4 KB, free: 265.1 MB)
15/01/19 21:33:31 INFO spark.ContextCleaner: Cleaned broadcast 202
Out of 24421 Predictions:
  20183 Correct!
   4238 Wrong!

Out of 5834 Salaries >50k : 2449 Correctly Predicted

Out of 5834 Salaries >50k : 3385 Wrongly Predicted

Out of 18587 Salaries  <=50k : 17734 Correctly Predicted

Out of 18587 Salaries  <=50k : 853 Wrongly Predicted

==================================================================
13. Result shows the Spark ignores worker that does not fullfill the requirement
    and just run on the nodes that fullfill the constraints

Application finished running 4 seconds earlier
```

# Appendix H

# Source Codes

## H.1 WordCount.java

```java
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;


public class WordCount
{

    public static class Map 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().replaceAll("\\p{Punct}|\\d","").toLowerCase();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens())
            {
                word.set(tokenizer.nextToken());
                output.collect(word,one);
            }
        }
    }
    public static class Reduce 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));
        }
    }
    public static void main(String[] args) throws Exception
    {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.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);
    }
}
```

## H.2 Dataset Accounting Codes

```
/*
 * Class Name: DataPeraparation.java
*/
/*
 * To change this license header, choose License Headers in Project
   Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package datapreparation;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

/**
 *
 * @author taqi
 */
public class DataPreparation {

    /**
     * @param args the command line arguments
     * @throws java.io.FileNotFoundException
     */
    public static void main(String[] args) throws
   FileNotFoundException{
         // TODO code application logic here
        int option;
        CreateFile file = new CreateFile();
        Scanner userInput;
        System.out.println("Menu:"
              + "\n1. Add Record Number"
              + "\n2. Count Number of Labels");
        System.out.println("\nChoose");
        userInput = new Scanner(System.in);
        option = userInput.nextInt();
        switch(option)
        {
            case 1:
            {
                 System.out.println("Enter full path of file:");
                 String path;
                 userInput = new Scanner(System.in);
                 path = userInput.nextLine();
                 userInput = new Scanner(new File(path));
                 file.openFile(path.replace(".txt",
    "")+"NumberedFile.txt");
                 int counter =1;
                 String line;
                 while (userInput.hasNextLine())
                 {
                     line = userInput.nextLine();
                     file.addRecord(line+", recNo-"+counter);
                     counter++;
                 }
                 file.closeFile();
            }
                break;
            case 2:
            {
                 String path;
                 userInput = new Scanner(System.in);
                 int moreLabel=0,lessLabel=0,unknownLabel=0;
                 String line;
                 System.out.println("Enter full path of file:");
                 path = userInput.nextLine();
                 userInput = new Scanner(new File(path));
                 while (userInput.hasNextLine())
                 {
                     line = userInput.nextLine();
                 if(line.contains(">50K"))
                     moreLabel++;
                 else if(line.contains("<=50"))
                     lessLabel++;
                 else
                     unknownLabel++;
                 }
                 System.out.println("Lines with >50K Label:
    "+moreLabel);
                 System.out.println("Lines with <=50K Label:
    "+lessLabel);
                 System.out.println("Lines with Unknown Label:
    "+unknownLabel);

            }
                 break;
            default:
            break;

        }

    }

}
```

## H.3 CreateFile.java

```java
/*
 * Class Name: CreateFile.java
 * Purpose: To create customized text files for the output of the
   program
 * Dependency: LogisticRegression.java needs this class to be able
 * to output its file and iteracts with test dataset
 */
package datapreparation;

/**
 *
 * @author taqi
 */

import java.util.Formatter;
public class CreateFile {
    private Formatter newFile;

    public void openFile(String fullPath){
        try {
            newFile = new Formatter(fullPath);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    public void addRecord(String line){
        try {
            newFile.format("%s", line+"\n");

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    public void closeFile(){
        try {
            newFile.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

## H.4 LinearRegression.java

```java
/*
 * Class Name: LogisticRegression.java
 * This is machine learning implementation of Logistic regression that
   enable you
 * to do classification on census dataset called Adult.
 * Given a vector of features for an individual person it will predict
   that whether
 * s/he is making more than 50k per year or less than or equal to that
   amount
 */
package org.fyp.sparkbasics;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.classification.LogisticRegressionModel;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
import org.apache.spark.mllib.feature.HashingTF;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.regression.LabeledPoint;

/**
 *
 * @author taqi
 */
public class LogisticRegression {

    public static void main(String[] args) throws
FileNotFoundException {
        //Connection of application to cluster through spark context
        SparkConf conf = new SparkConf().setAppName("Test Logistic
Regression").setMaster("local");
        JavaSparkContext sc = new JavaSparkContext(conf);

        JavaRDD<String> less =
sc.textFile("hdfs://localhost:9000/lessOrEqual50K/lessThan50K.txt");
        JavaRDD<String> more =
sc.textFile("hdfs://localhost:9000/moreThan50K/moreThan50K.txt");

        // Create a HashingTF instance to map people text to vectors
of 10000 features.
        final HashingTF tf = new HashingTF(10000);

        // Create LabeledPoint datasets for moreThan50KExample (1) and
lessThan50KExamples (0)
        JavaRDD<LabeledPoint> moreThan50KExamples = more.map(new
Function<String, LabeledPoint>() {
            public LabeledPoint call(String email) {
                return new LabeledPoint(1,
tf.transform(Arrays.asList(email.split(","))));
            }
        });
        JavaRDD<LabeledPoint> lessThan50KExamples = less.map(new
Function<String, LabeledPoint>() {
            public LabeledPoint call(String email) {
                return new LabeledPoint(0,
tf.transform(Arrays.asList(email.split(","))));
            }
        });

```

```java
        JavaRDD<LabeledPoint> trainingData = moreThan50KExamples.union(lessThan50KExamples);
        trainingData.cache(); // Cache since Logistic Regression is an iterative algorithm.

        // Run Logistic Regression using the SGD algorithm.
        LogisticRegressionModel model = new LogisticRegressionWithSGD().run(trainingData.rdd());

        // LogisticRegression on a positive example (spam) and a negative one (normal).
        Vector test;

      // System.out.println("Prediction " + model.predict(test));
        String path = "/home/taqi/Dropbox/MMU10/FYP/Notes/DatasetAdult/adult.test";

        Scanner readFromAdultTest;
        readFromAdultTest = new Scanner(new File(path));
        CreateFile prediction;
        prediction = new CreateFile();
        prediction.openFile("prediction.txt");
        String line;
        String lineWithoutLabel;
        int counter = 0;
        int correctPredictions = 0;
        int wrongPredictions = 0;
        while (readFromAdultTest.hasNextLine()) {
            line = readFromAdultTest.nextLine();
            if (line.contains("<=50K")) {
                lineWithoutLabel = line.replace(", <=50K.", "");
                test = tf.transform(Arrays.asList(lineWithoutLabel.split(",")));
                if (model.predict(test) == 0.0) {
                    prediction.addRecord(line + " Prediction is <=50K , Correct");
                    correctPredictions++;
                } else {
                    prediction.addRecord(line + " Prediction is >50K , Wrong");
                    wrongPredictions++;
                }
                counter++;
            } else if (line.contains(">50K")) {
                lineWithoutLabel = line.replace(", >50K.", "");
                test = tf.transform(Arrays.asList(lineWithoutLabel.split(",")));
                if (model.predict(test) == 1.0) {
                    prediction.addRecord(line + " Prediction is >50K , Correct");
                    correctPredictions++;
                } else {
                    prediction.addRecord(line + " Prediction is <=50K , Wrong");
                    wrongPredictions++;
                }
                counter++;
            }

        }
        System.out.println("Out of " + counter + " Predictions: \n\t" + correctPredictions + " Correct! \n\t " + wrongPredictions + " Wrong!\n");
        double Error = (double) correctPredictions / (double) counter;
        System.out.printf("%.5f%s", Error, "\n");
    }
}
```

# Appendix I

# Sample of Adult Dataset

In this appendix we show the sample of dataset we obtained from UCI repository and datasets we generated out of them for our case studies. For complete information about this data set please refer to the following URL:
`https://archive.ics.uci.edu/ml/datasets/Adult`
It has 14 features separated by comma and each record by a new line.
Features' details are as below:

1. age: continuous.
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

7. occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
8. relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. sex: Female, Male.
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

1. The following two files are the sample of dataset we downloaded from UCI machine learning repository with no changes on them.
   Note: Each line on the left side of the box represent one line in the file

```
1 ============================================================
2 Sample 1. The following is a sample data taken from adult.data
3 ============================================================
4 39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical,
      Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
5 50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
      Exec-managerial, Husband, White, Male, 0, 0, 13,
      United-States, <=50K
```

```
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners,
    Not-in-family, White, Male, 0, 0, 40, United-States, <=50K
53, Private, 234721, 11th, 7, Married-civ-spouse,
    Handlers-cleaners, Husband, Black, Male, 0, 0, 40,
    United-States, <=50K
28, Private, 338409, Bachelors, 13, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K
37, Private, 284582, Masters, 14, Married-civ-spouse,
    Exec-managerial, Wife, White, Female, 0, 0, 40, United-States,
    <=50K

....
UPTO 32561 RECORD LINES
```

```
============================================================
The following is a sample data taken from adult.test
============================================================
25, Private, 226802, 11th, 7, Never-married, Machine-op-inspct,
    Own-child, Black, Male, 0, 0, 40, United-States, <=50K.
38, Private, 89814, HS-grad, 9, Married-civ-spouse,
    Farming-fishing, Husband, White, Male, 0, 0, 50,
    United-States, <=50K.
28, Local-gov, 336951, Assoc-acdm, 12, Married-civ-spouse,
    Protective-serv, Husband, White, Male, 0, 0, 40,
    United-States, >50K.
44, Private, 160323, Some-college, 10, Married-civ-spouse,
    Machine-op-inspct, Husband, Black, Male, 7688, 0, 40,
    United-States, >50K.
18, ?, 103497, Some-college, 10, Never-married, ?, Own-child,
    White, Female, 0, 0, 30, United-States, <=50K.
34, Private, 198693, 10th, 6, Never-married, Other-service,
    Not-in-family, White, Male, 0, 0, 30, United-States, <=50K.
29, ?, 227026, HS-grad, 9, Never-married, ?, Unmarried, Black,
    Male, 0, 0, 40, United-States, <=50K.
```

```
63, Self-emp-not-inc, 104626, Prof-school, 15,
    Married-civ-spouse, Prof-specialty, Husband, White, Male,
    3103, 0, 32, United-States, >50K.
24, Private, 369667, Some-college, 10, Never-married,
    Other-service, Unmarried, White, Female, 0, 0, 40,
    United-States, <=50K.
55, Private, 104996, 7th-8th, 4, Married-civ-spouse,
    Craft-repair, Husband, White, Male, 0, 0, 10, United-States,
    <=50K.

....
UPTO 24421 RECORD LINES
```

2. The following listing is the sample of the text files we create from the above two files.

```
============================================================
The following is a sample data taken from rawData.txt
rawData.txt is a combination of adult.data and adult.test
============================================================
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical,
    Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 13,
    United-States, <=50K
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners,
    Not-in-family, White, Male, 0, 0, 40, United-States, <=50K
53, Private, 234721, 11th, 7, Married-civ-spouse,
    Handlers-cleaners, Husband, Black, Male, 0, 0, 40,
    United-States, <=50K
28, Private, 338409, Bachelors, 13, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K
```

```
37, Private, 284582, Masters, 14, Married-civ-spouse,
    Exec-managerial, Wife, White, Female, 0, 0, 40, United-States,
    <=50K

....
UP TO 48842 RECORD LINES
```

3. The following listing shows the raw data that we showed in the number 2 but the difference is that we added record (line) number at the end of each line.

```
============================================================
The following is a sample data taken from numberedRaw.txt
numberedRaw.txt is basically rawData.txt but for each line
we added recNo-# which is record number of that line
============================================================
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical,
    Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K,
    recNo-1
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 13,
    United-States, <=50K, recNo-2
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners,
    Not-in-family, White, Male, 0, 0, 40, United-States, <=50K,
    recNo-3
53, Private, 234721, 11th, 7, Married-civ-spouse,
    Handlers-cleaners, Husband, Black, Male, 0, 0, 40,
    United-States, <=50K, recNo-4
28, Private, 338409, Bachelors, 13, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K,
    recNo-5
37, Private, 284582, Masters, 14, Married-civ-spouse,
    Exec-managerial, Wife, White, Female, 0, 0, 40, United-States,
    <=50K, recNo-6
```

```
49, Private, 160187, 9th, 5, Married-spouse-absent,
    Other-service, Not-in-family, Black, Female, 0, 0, 16,
    Jamaica, <=50K, recNo-7

....
UPTO 48842 RECORD LINES
```

4. After adding number to end of our raw data we split it from half and saved output in two files:
   A.train.txt which is used for training the classifier

```
===========================================================
This is the sample data taken from train.txt
===========================================================
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical,
    Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K,
    recNo-1
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 13,
    United-States, <=50K, recNo-2
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners,
    Not-in-family, White, Male, 0, 0, 40, United-States, <=50K,
    recNo-3
53, Private, 234721, 11th, 7, Married-civ-spouse,
    Handlers-cleaners, Husband, Black, Male, 0, 0, 40,
    United-States, <=50K, recNo-4
28, Private, 338409, Bachelors, 13, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K,
    recNo-5

.....
UP TO 24421 RECORD LINES
```

B.test.txt which is the records that we do prediction for them.

```
============================================================
Sample data taken from test.txt
============================================================
50, Private, 99925, HS-grad, 9, Married-civ-spouse, Adm-clerical,
    Husband, White, Male, 0, 0, 32, United-States, <=50K,
    recNo-24422
58, Private, 227800, 1st-4th, 2, Separated, Farming-fishing,
    Not-in-family, Black, Male, 0, 0, 50, United-States, <=50K,
    recNo-24423
55, State-gov, 111130, Assoc-acdm, 12, Divorced, Adm-clerical,
    Own-child, Asian-Pac-Islander, Male, 0, 0, 40, United-States,
    <=50K, recNo-24424
29, Private, 100764, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 45,
    United-States, >50K, recNo-24425
47, Private, 275095, Some-college, 10, Divorced,
    Machine-op-inspct, Not-in-family, White, Female, 0, 0, 40,
    United-States, <=50K, recNo-24426
39, Private, 147500, HS-grad, 9, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, United-States,
    <=50K, recNo-24427
63, Local-gov, 150079, HS-grad, 9, Married-civ-spouse,
    Adm-clerical, Wife, White, Female, 0, 0, 35, United-States,
    >50K, recNo-24428

....
UP TO 24421 RECORD LINES
```

5. Our program need that train to be separated based on the label, hence the following is the sample of train.txt which has been divided based on the label.

A. The following records are sample of data with label more than 50K

```
===================================================
Sample data taken from moreThank50kTrain.txt
===================================================
52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 45, United-States
31, Private, 45781, Masters, 14, Never-married, Prof-specialty,
    Not-in-family, White, Female, 14084, 0, 50, United-States
42, Private, 159449, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 5178, 0, 40,
    United-States
37, Private, 280464, Some-college, 10, Married-civ-spouse,
    Exec-managerial, Husband, Black, Male, 0, 0, 80, United-States
30, State-gov, 141297, Bachelors, 13, Married-civ-spouse,
    Prof-specialty, Husband, Asian-Pac-Islander, Male, 0, 0, 40,
    India
40, Private, 121772, Assoc-voc, 11, Married-civ-spouse,
    Craft-repair, Husband, Asian-Pac-Islander, Male, 0, 0, 40, ?

....
UPTO 5834 RECORD LINES
```

B. The following records are sample of data with label less than or equal 50K

```
========================================================
Sample data taken from lessThan50kTrain.txt
========================================================
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical,
    Not-in-family, White, Male, 2174, 0, 40, United-States
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 13, United-States
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners,
    Not-in-family, White, Male, 0, 0, 40, United-States
53, Private, 234721, 11th, 7, Married-civ-spouse,
    Handlers-cleaners, Husband, Black, Male, 0, 0, 40,
    United-States

....
UPTO 18568 RECORD LINES
```

6. After running the classification we generate a prediction.txt file that contains all the predictions with their predicted label and actual label.

```
============================================================
Sample data taken from prediction.txt
============================================================
50, Private, 99925, HS-grad, 9, Married-civ-spouse, Adm-clerical,
    Husband, White, Male, 0, 0, 32, United-States, <=50K,
    recNo-24422, Prediction is <=50K , Correct
58, Private, 227800, 1st-4th, 2, Separated, Farming-fishing,
    Not-in-family, Black, Male, 0, 0, 50, United-States, <=50K,
    recNo-24423, Prediction is <=50K , Correct
55, State-gov, 111130, Assoc-acdm, 12, Divorced, Adm-clerical,
    Own-child, Asian-Pac-Islander, Male, 0, 0, 40, United-States,
    <=50K, recNo-24424, Prediction is <=50K , Correct
29, Private, 100764, Bachelors, 13, Married-civ-spouse,
    Exec-managerial, Husband, White, Male, 0, 0, 45,
    United-States, >50K, recNo-24425, Prediction is >50K , Correct
47, Private, 275095, Some-college, 10, Divorced,
    Machine-op-inspct, Not-in-family, White, Female, 0, 0, 40,
    United-States, <=50K, recNo-24426, Prediction is <=50K ,
    Correct
39, Private, 147500, HS-grad, 9, Married-civ-spouse,
    Prof-specialty, Wife, Black, Female, 0, 0, 40, United-States,
    <=50K, recNo-24427, Prediction is <=50K , Correct

....
UP TO 24421 RECORD LINES
```