Bachelor Thesis Project

ANALYSIS OF FLIGHTS DATASET AND PREDICTION OF DELAYS USING A HADOOP CLUSTER

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF B.E (COMPUTER ENGINEERING)

SUBMITTED BY:

Vasu Sharma (385/CO/13)

Ved Prakash Sachdeva (386/CO/13)

Vidur Katyal (387/CO/13)

GUIDED BY:

Pinaki Chakraborty

(Assistant Professor, Division of Computer Engineering)

DIVISION OF COMPUTER ENGINEERING NETAJI SUBHAS INSTITUTE OF TECHNOLOGY UNIVERSITY OF DELHI 2016-2017

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our advisor Dr Pinaki Chakraborty for giving us the opportunity to work under his able supervision. He has been a pillar of strength throughout this research, showing us the way from the very beginning. The frequent brainstorming sessions have helped us bring out the best and have made us always feel excited about the work we are doing. His encouragement in times of distress and anxiety; patience in hearing our problems; rigorous reviewing to make sure that everything falls in place; and the immense knowledge that he possesses have helped us carry this project forward in the right spirits. Without his invaluable and continuous contributions, this project would not have been completed.

We are also indebted towards the faculty of NSIT, whose guidance and teaching for the past four years has helped us in understanding the theory and practices of Computer Engineering. The progress in our Bachelor Thesis Project is a reflection of the progression of our education in the four years that we have spent at NSIT.

We also thank our parents and families who always supported us unconditionally in our thick and thin. Their belief in our capabilities always pushed us to give our best in this project.

नेताजी सुभाष प्रौद्योगिकी संस्थान

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

(An Institution of Govt. of NCT of Delhi-Formerly, Delhi Institute of Technology)

Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi - 110 078
Telphone : 25099050; Fax : 25099025, 25099022 Website : http://www.nsit.ac.in

DECLARATION

This is to certify that the project entitled, **“Analysis of Flights Dataset and Prediction of Delays Using a Hadoop Cluster”** by **Vasu Sharma (385/CO/13)**, **Ved Prakash Sachdeva (386/CO/13)** and **Vidur Katyal (387/CO/13)** is a record of the bonafide work carried out by us, in the Division of Computer Engineering, Netaji Subhas Institute of Technology, University of Delhi, New Delhi. The thesis is being submitted in partial fulfilment of requirements for the Degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2016-2017.

The results presented in this thesis have not been submitted to any other university or institute in any form for the award of any other degree or diploma.

Vasu Sharma
(385/CO/13)

Ved Prakash Sachdeva
(386/CO/13)

Vidur Katyal
(387/CO/13)

नेताजी सुभाष प्रौद्योगिकी संस्थान

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

(An Institution of Govt. of NCT of Delhi-Formerly, Delhi Institute of Technology)

Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi - 110 078
Telphone : 25099050; Fax : 25099025, 25099022 Website : http://www.nsit.ac.in

CERTIFICATE

This is to certify that the project entitled, **"Analysis of Flights Dataset and Prediction of Delays Using a Hadoop Cluster"** by **Vasu Sharma (385/CO/13)**, **Ved Prakash Sachdeva (386/CO/13)** and **Vidur Katyal (387/CO/13)** is a record of the bonafide work carried out by them, in the Division of Computer Engineering, Netaji Subhas Institute of Technology, University of Delhi, New Delhi, under my supervision and guidance in partial fulfilment of requirements for the award of the Degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2016-2017.

The results presented in this thesis have not been submitted to any other university or institute in any form for the award of any other degree or diploma.

Dr Pinaki Chakraborty
Assistant Professor
Division of Computer Engineering
Netaji Subhas Institute of Technology (NSIT)
Azad Hind Fauj Marg
Sector-3, Dwarka
New Delhi

TABLE OF CONTENTS

ACKNOWLEDGEMENT..2

DECLARATION..3

CERTIFICATE..4

LIST OF FIGURES...6

LIST OF TABLES..8

ABSTRACT……………………………………………………………………………..9

INTRODUCTION TO PROJECT...10

INTRODUCTION...11

BIG DATA……………………………………………………………………11

APACHE HADOOP…………………………………………………………….14

APACHE SPARK………………………………………………………………15

PROJECT REQUIREMENTS...16

SYSTEM REQUIREMENTS..16

SOFTWARE REQUIREMENTS..16

PROJECT APPROACH..18

OBJECTIVES...20

RESEARCH..21

THEORETICAL BACKGROUND...22

MAPREDUCE…………………………………………………….....................22

HDFS……………………………………………………………………………24

YARN…………………………………………………….................................…...27

APACHE SPARK……………………………………………………...............…29

SUPPORT VECTOR MACHINES……………………………………..............33

DECISION TREE………………………………………………….................35

IMPLEMENTATION DETAILS...37

SETTING UP HADOOP CLUSTER……………………………………….................38

DATA GATHERING AND

AGGREGATION……………………………………..........40

DATA EXPLORATION AND

VISUALISATION…………………………….........44

SETTING UP APACHE SPARK………………………………..………………...57

TRAINING THE PREDICTION MODEL…………………………………….............59

CONCLUSION...65

FUTURE WORK...66

REFERENCES..67

LIST OF FIGURES

Figure 3.1: Pseudo Code for MapReduce…………………………………………………........22

Figure 3.2: Overview of a MapReduce Job…………………………………………………….24

Figure 3.3: HDFS Architecture………...…………………………………………………….25

Figure 3.4: YARN Structure…………….………………………………………………....27

Figure 3.5: YARN Cluster

Management…………………...……………………….……...28

Figure 3.6: Iterative Job Execution using Spark's

RDD……………………………...………31

Figure 3.7: Interactive Job Execution using Spark's RDD……………………………………...31

Figure 3.8: Components of Spark……..……………………………………...……...32

Figure 3.9: Classification using SVM….………………………………...……...……...34

Figure 3.10: Classification using Decision Tree...…………………………………...……....35

Figure 4.1: HDFS Datanode Information...……………………………………...……………39

Figure 4.2: Sample Code for Mapper...………………………………….....…………………42

Figure 4.3: Sample Code for Reducer...……………………………….....………………...42

Figure 4.4: Cluster Metrics during Execution of Map Phase………………………...……...43

Figure 4.5: Cluster Metrics During Execution of Reduce Phase……...………………...……...43

Figure 4.6: Histogram depicting Departure Delay Distribution..………………...……………44

Figure 4.7: Histogram depicting Arrival Delay Distribution…………………...…………..45

Figure 4.8: Bar Chart depicting Average Departure Delay per

Month...…………..……..……..47

Figure 4.9: Bar Chart depicting Average Arrival Delay per Month...……………….....………..47

Figure 4.10: Bar Chart depicting Average Departure Delay per Hour of Day…..………………....49

Figure 4.11: Bar Chart depicting Average Arrival Delay per Hour of Day………...…………..49

Figure 4.12: Bar Chart depicting Average Departure Delay per Day of Week...……......……....51

Figure 4.13: Bar Chart depicting Average Arrival Delay per Day of Week...……......…………51

Figure 4.14: Bar Chart depicting Average Departure Delay per Carrier...……......……………52

Figure 4.15: Bar Chart depicting Average Arrival Delay per Carrier...……......………...…......53

Figure 4.16: Line Chart depicting percentage of flights delayed per day for ORD in 2008……55

Figure 4.17: Line Chart depicting percentage of flights delayed per day for SFO in 2004…….55

Figure 4.18: Line Chart depicting percentage of flights delayed per day for JFK in 2005…….56

Figure 4.19: Spark DAG Visualisation………...……………………………………...……....61

Figure 4.20: Metrics for a Spark job…………...……………………………………...……....61

Figure 4.21: Performance Metrics for SVM...…………………………………...……………62

Figure 4.22: Performance Metrics for Decision Tree……………………………....………..62

Figure 4.23: Performance Metrics for SVM...…………………………………...……….....63

Figure 4.24: Performance Metrics for Decision Tree……………………………....………..64

LIST OF TABLES

Table 4.1: Columns in Flights Dataset...…………………………..……………..…....…….40

Table 4.2: Confusion Matrix for SVM...………………………..……………..…....…….62

Table 4.3: Confusion Matrix for Decision Tree...………………………………..…....…….62

Table 4.4: Confusion Matrix for SVM...……………………………………....……....…….63

Table 4.5: Confusion Matrix for Decision Tree...…………………….……………....……...64

ABSTRACT

Every year approximately 20%[27] of airline flights are delayed or cancelled, resulting in significant costs to both travellers and airlines. Flight delays have a negative effect on airlines, airports and passengers and the economy. Even a 10% reduction in flight delays could increase the US net welfare by $17.6 billion and a 30% reduction can put this figure at $38.5 billion[28]. Flight delays have been attributed to several causes such as weather conditions, airport congestion, airspace congestion, use of smaller aircraft by airlines, etc. Their prediction is crucial during the decision-making process for all players of commercial aviation. The datasets we used are the Airline On-Time Performance Data from the Federal Aviation Administration (FAA) released by Statistical Computing and the Surface Airways Weather Data from the National Climatic Data Center (NCDC).

The aim of this thesis is to use exploratory analysis and to develop machine learning models to predict airline's departure and arrival delays using a Hadoop Cluster. Apache Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. With Apache Spark's MLlib library's vast collection of classification, regression and tree based models and Spark's ability to work on a Hadoop Cluster, we will build these prediction models.

INTRODUCTION TO PROJECT

INTRODUCTION

We live in a world that is driven by Data. Data intensive applications are on the rise. Since the communication paradigm of the Internet is sufficiently open and powerful, the World Wide Web, in the past decade or so, has been adopted as the ideal platform for developing data intensive applications. Data intensive applications like data mining and web indexing need to access ever expanding data sets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example leverages the MapReduce model to process approximately twenty petabytes of data per day in a parallel scenario[1]. The MapReduce programming framework can simplify the complexity by running parallel data processing functions across multiple computing nodes in a cluster as scalable MapReduce helps programmers to distribute programs across nodes as they as executed in parallel. MapReduce automatically gathers results from the multiple nodes and returns a single result or set. More importantly, MapReduce platforms offer fault tolerance that is entirely transparent to programmers. This makes MapReduce a practical and attractive programming model for parallel data processing in high performance cluster computing environments.

Big Data

Big data is a blanket term for the non-traditional strategies and technologies needed to gather, organize, process, and gather insights from large datasets. While the problem of working with data that exceeds the computing power or storage of a single computer is not new, the pervasiveness, scale, and value of this type of computing has greatly expanded in recent years.

An exact definition of "big data" is difficult to nail down because projects, vendors, practitioners, and business professionals use it quite differently. With that in mind, generally speaking, big data is:

- large datasets
- the category of computing strategies and technologies that are used to handle large datasets

In this context, "large dataset" means a dataset too large to reasonably process or store with traditional tooling or on a single computer. This means that the common scale of big datasets is constantly shifting and may vary significantly from organization to organization.

The basic requirements for working with big data are the same as the requirements for working with datasets of any size. However, the massive scale, the speed of ingesting and processing, and the characteristics of the data that must be dealt with at each stage of the process present significant new challenges when designing solutions. The goal of most big data systems is to surface insights and connections from large volumes of heterogeneous data that would not be possible using conventional methods.

In 2001, Gartner's Doug Laney[2] first presented what became known as the "three Vs of big data" to describe some of the characteristics that make big data different from other data processing:

Volume

The sheer scale of the information processed helps define big data systems. These datasets can be orders of magnitude larger than traditional datasets, which demands more thought at each stage of the processing and storage life cycle.

Often, because the work requirements exceed the capabilities of a single computer, this becomes a challenge of pooling, allocating, and coordinating resources from groups of computers. Cluster management and algorithms capable of breaking tasks into smaller pieces become increasingly important.

Velocity

Another way in which big data differs significantly from other data systems is the speed that information moves through the system. Data is frequently flowing into the system from multiple sources and is often expected to be processed in real time to gain insights and update the current understanding of the system.

This focus on near instant feedback has driven many big data practitioners away from a batch-oriented approach and closer to a real-time streaming system. Data is constantly being added, massaged, processed, and analyzed in order to keep up with the influx of new information and to surface valuable information early when it is most relevant. These ideas require robust systems with highly available components to guard against failures along the data pipeline.

Variety

Big data problems are often unique because of the wide range of both the sources being processed and their relative quality.

Data can be ingested from internal systems like application and server logs, from social media feeds and other external APIs, from physical device sensors, and from other providers. Big data seeks to handle potentially useful data regardless of where it's coming from by consolidating all information into a single system.

The formats and types of media can vary significantly as well. Rich media like images, video files, and audio recordings are ingested alongside text files, structured logs, etc. While more traditional data processing systems might expect data to enter the pipeline already labeled, formatted, and organized, big data systems usually accept and store data closer to its raw state. Ideally, any transformations or changes to the raw data will happen in memory at the time of processing.

Apache Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Apache Spark

Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. It is an open-source cluster-computing framework, originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

PROJECT REQUIREMENTS

System Requirements

1. Operating System: Linux/Unix.

   This project was performed on machines running Ubuntu 16.04 (Xenial Xerus).

Software Requirements

1. **Python 2.7 or later**: Python[8] is a widely used general-purpose, multi-paradigm, dynamically-typed high-level programming language. Python was chosen for this project for its ability to allow rapid prototyping of applications and for its wide support-base.
2. **Scala 2.12 or later**: Scala[9] is a general-purpose programming language providing support for functional programming and a strong static type system. Scala was chosen for this project due to presence of well-defined Scala APIs in Apache Spark.
3. **Apache Hadoop 2.7.3 or later**: Hadoop is an open-source software framework used for distributed storage and processing of dataset of big data using the MapReduce programming model. It consists of computer clusters built from commodity hardware
4. **Apache Spark 2.11.8 or later**: Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. It is an open-source cluster-computing framework managed by the Apache Software Foundation.

5. **Python Libraries used:**

   a. **Matplotlib:** Matplotlib[10] is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. It can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, etc.

   b. **ggplot:** ggplot[11] is a plotting system for Python based on R's ggplot2 and the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

   c. **NumPy:** NumPy[12] is a general-purpose array-processing package designed to efficiently manipulate large multi-dimensional arrays of arbitrary records without sacrificing too much speed for small multi-dimensional arrays.

   d. **pandas:** pandas[13] is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

PROJECT APPROACH

- **Topic Research and Selection:** We read various journals, articles and technological blogs and held discussions with our faculty incharge to identify the topic where there was scope for research and implementation; and finally decided to work on state of the art big data processing technology Hadoop.
- **Research:** We researched various tech blogs to find out various use cases which can be solved using Hadoop and some publicly available datasets to use in our project. We came across a GitHub repository[3] containing links to various publicly available datasets.
- **Dataset Collection and Preparation:** We finalised the airlines on-time performance dataset obtained from Statistical Computing[4]. This dataset was originally released by RITA and was used by the Statistical Computing for their competition the Data Expo '09. It consists of flight arrival and departure details for all commercial flights in the USA, from October 1987 to December 2008. There are nearly 120 million records in total taking up a total of 12 gigabytes.
- **Setting up the Apache Hadoop Cluster:** We tried to follow the Apache Hadoop guide to setup the Hadoop Cluster. With assistance from various technology blogs, we were able to setup a single node as well as a multi node cluster. The multi node cluster was setup in Graphics Lab (Room No 103/V) at NSIT, New Delhi.
- **Aggregating data for initial research:** We used MapReduce to aggregate the flights data. MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. We referred to various online resources to understand MapReduce model and applying it

on the Apache Hadoop Cluster. We used the Hadoop Streaming library to run MapReduce jobs written in Python programming language.

- **Analysing the Results obtained:** The results obtained from the MapReduce jobs were plotted using the Python libraries ggplot and matplotlib to properly visualise the data in order to select the parameters for the prediction model.
- **Training the Prediction model:** The training of the prediction model was done using Apache Spark. Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. We used Spark's MLlib library to train the prediction model.

OBJECTIVES

The objective of this thesis is to use Big Data analysis tools such as Apache Hadoop to analyse the dataset of commercial flights of USA, with the aim of building a prediction model to predict delays in airlines.

This thesis focuses on three areas:

1. Comprehensive study of the Apache Hadoop's framework
2. Systematic analysis of the flights dataset obtained
3. Building a prediction model to predict delays in flights

RESEARCH

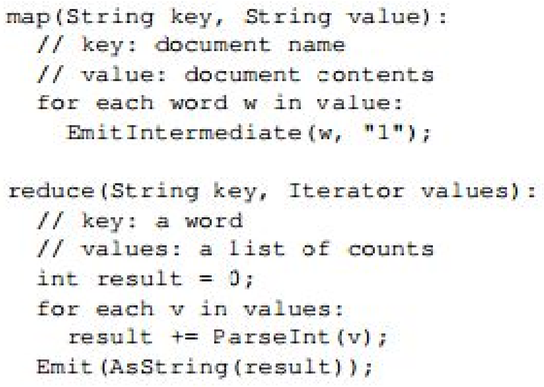
THEORETICAL BACKGROUND

Background of MapReduce

MapReduce[5] is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown below.

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce.

Example: Consider the problem of counting the number of occurrences of each word in a large collection of documents. The user would write code similar to the following pseudo-code:



```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

FIGURE 3.1: PSEUDO CODE FOR MAPREDUCE

Here, the map function emits each word plus an associated count of occurrences (just ‘1’ in this simple example). The reduce function sums together all counts emitted for a particular word.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides![5]
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** : The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.

- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

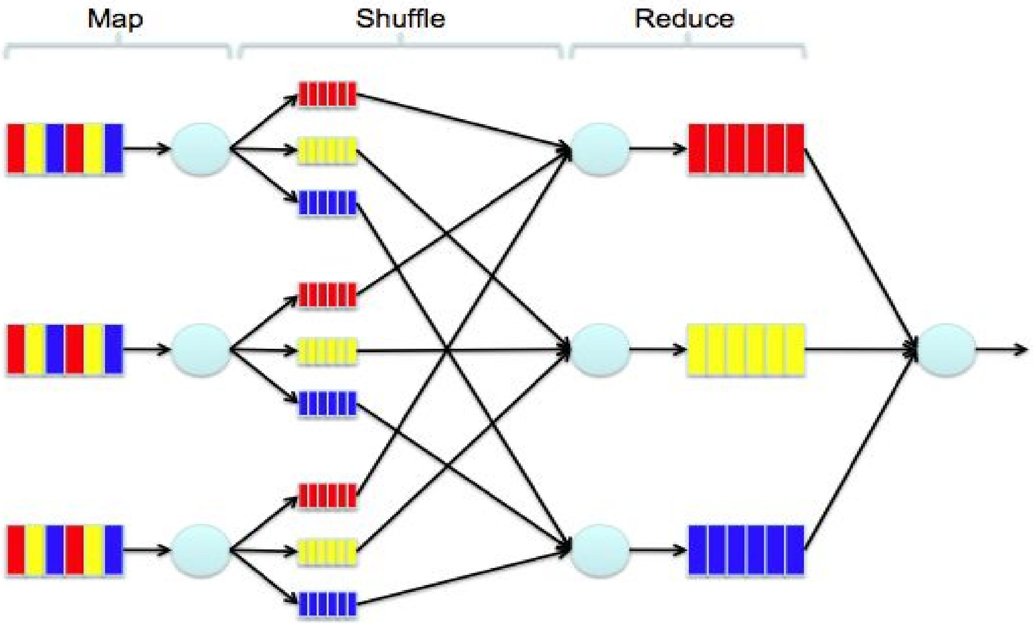


FIGURE 3.2: OVERVIEW OF A MAPREDUCE JOB

Background of HDFS

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.[6]

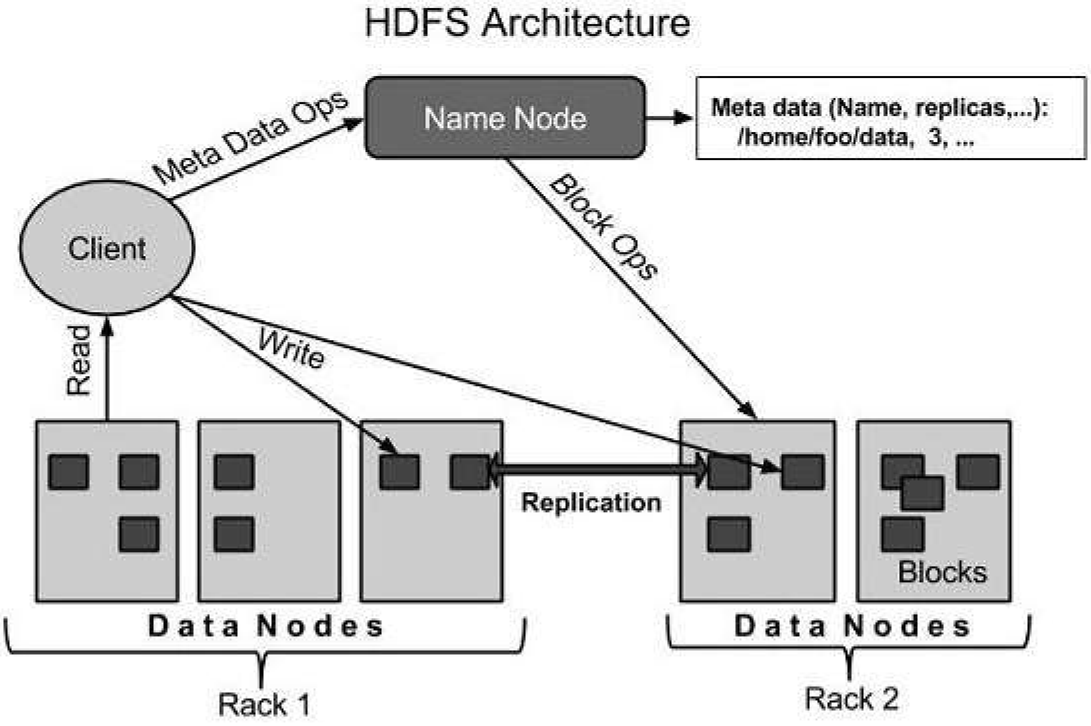


FIGURE 3.3: HDFS ARCHITECTURE

HDFS follows the master-slave architecture and it has the following elements:-

NameNode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.

- Regulates client’s access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

DataNode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 128MB, but it can be modified as per need in the HDFS configuration.

Background of YARN

YARN (**Y**et **A**nother **R**esource **N**egotiator)[22] is a new component added in Hadoop 2.0

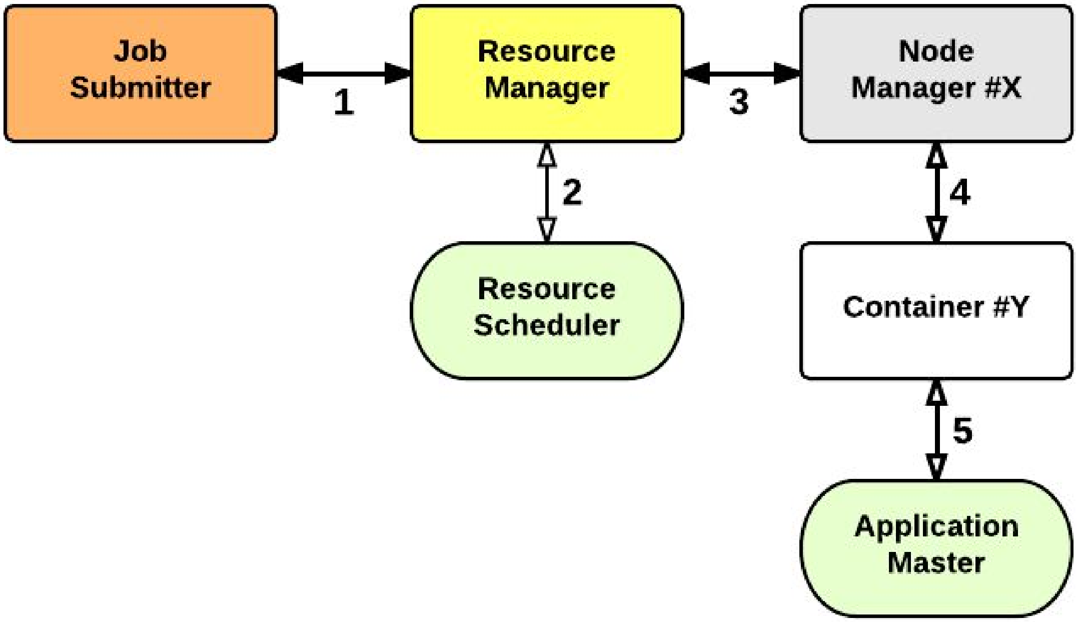


FIGURE 3.4: YARN STRUCTURE

In Hadoop 2.0 a new layer was introduced between HDFS and MapReduce.

The YARN framework is responsible for doing Cluster Resource Management.

Cluster Resource Management:

Cluster resource management means managing the resources of the Hadoop Clusters. In this case, resources mean Memory, CPU, etc.

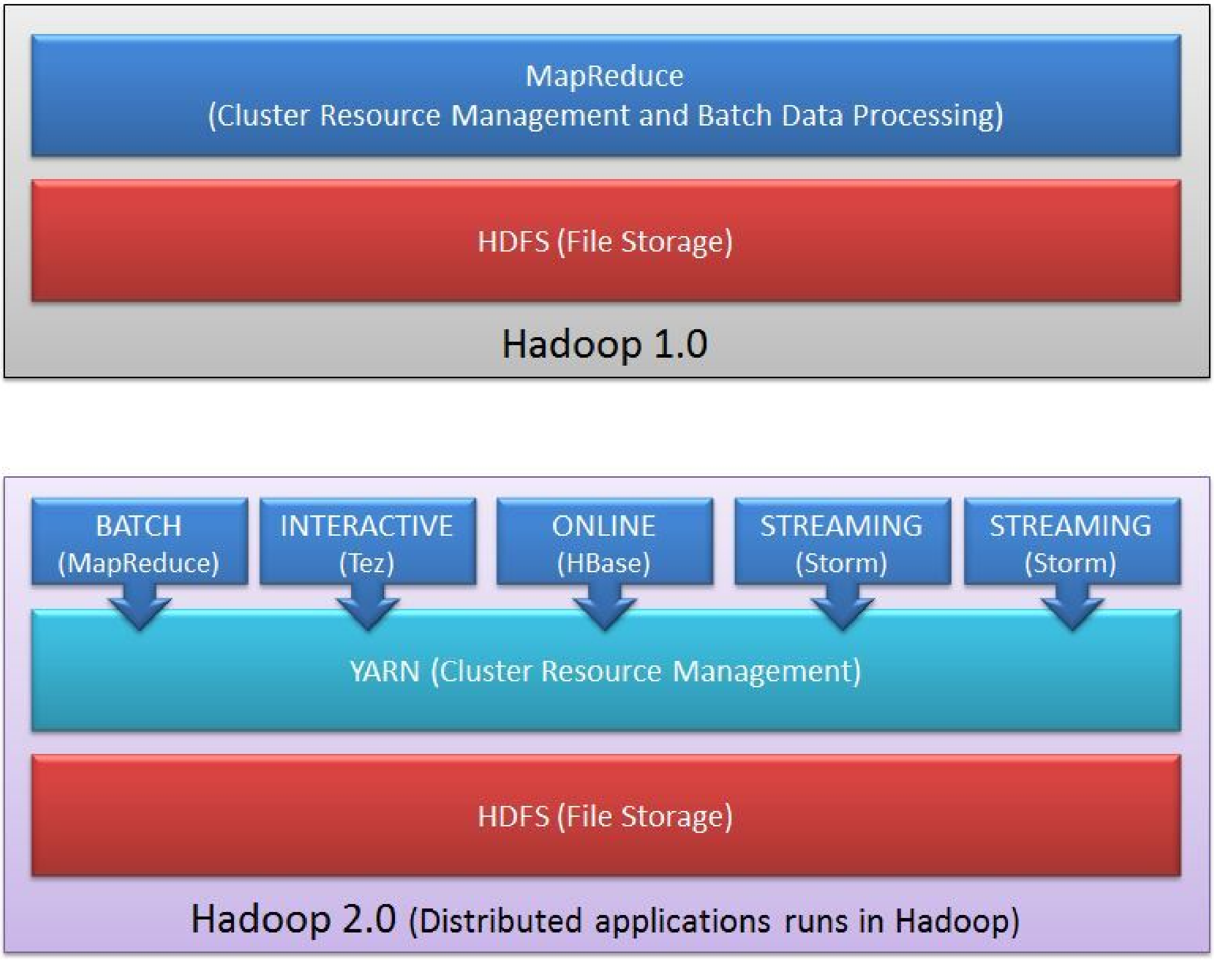

MapReduce
(Cluster Resource Management and Batch Data Processing)
HDFS (File Storage)
Hadoop 1.0
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm)
STREAMING (Storm)
YARN (Cluster Resource Management)
HDFS (File Storage)
Hadoop 2.0 (Distributed applications runs in Hadoop)

FIGURE 3.5: YARN CLUSTER MANAGEMENT

YARN[14] took over the task of cluster management from MapReduce and MapReduce is streamlined to perform Data Processing only. YARN has central resource manager component which manages resources and allocates the resources to the application. Multiple applications can run on Hadoop via YARN and all application could share common resource management.

Features of YARN:

It is backward compatible: This means that existing MapReduce job can run on Hadoop 2.0 without any change.

Removal of JobTracker and TaskTracker: YARN splits the two major functionalities of the JobTracker i.e. resource management and job scheduling/monitoring into 2 separate daemons (components).

- Resource Manager
- Node Manager(node specific)

Central Resource Manager and node specific Node Manager together constitutes YARN.

Background of Apache Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its **in-memory cluster computing** that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.[7]

Features of Apache Spark

- **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- **Advanced Analytics** − Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD)[19] is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

There are two ways to create RDDs − **parallelizing** an existing collection in the driver program, or by **referencing** a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Iterative and **Interactive** applications often requires faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to replication, serialization, and disk IO. Regarding storage system, most of the Hadoop applications, spend more than 90% of the time doing HDFS read-write operations.

Iterative Operations on Spark RDD

The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

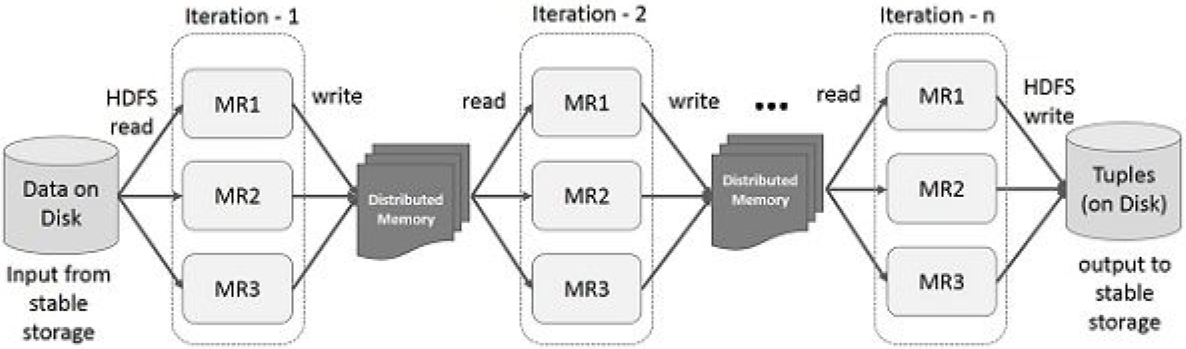


FIGURE 3.6: ITERATIVE JOB EXECUTION USING SPARK'S RDD

Interactive Operations on Spark RDD

This illustration below shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.

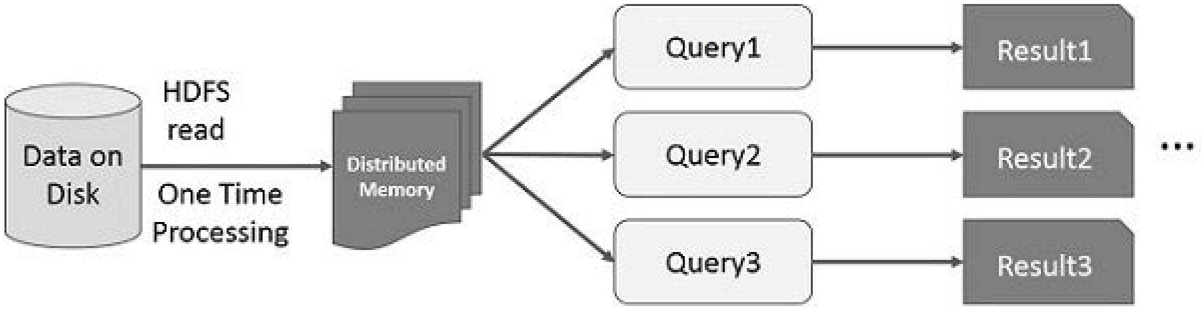


FIGURE 3.7: INTERACTIVE JOB EXECUTION USING SPARK'S RDD

COMPONENTS OF APACHE SPARK

The following illustration depicts the different components of Spark:-

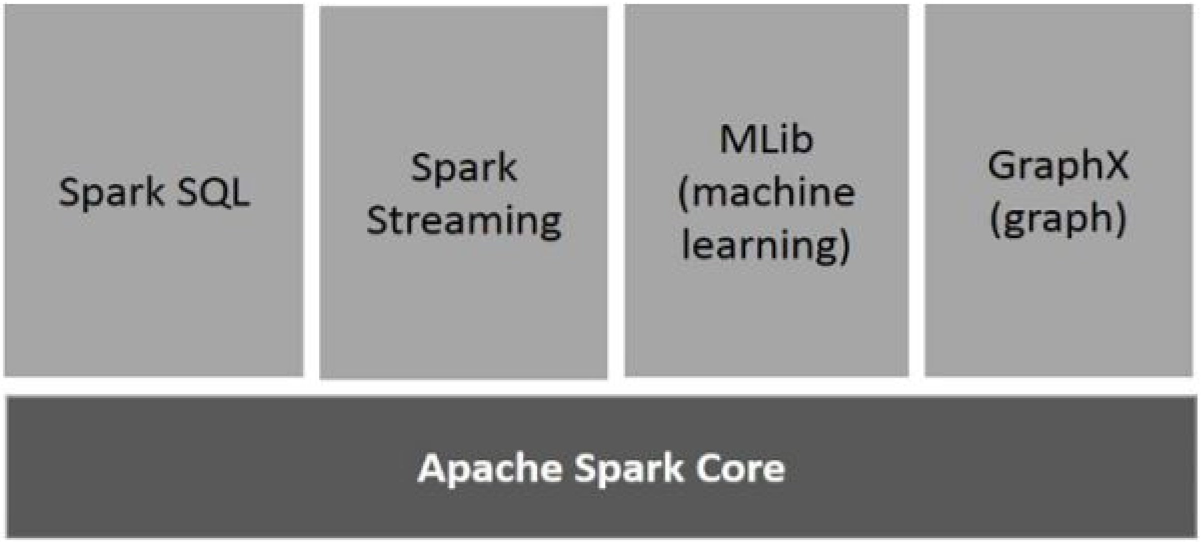


FIGURE 3.8: COMPONENTS OF SPARK

Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD[19] (Resilient Distributed Datasets) transformations on those mini-batches of data.

MLlib (Machine Learning Library)

MLlib[15] is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of **Apache Mahout** (before Mahout gained a Spark interface).

GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

SUPPORT VECTOR MACHINES

Support vector machines (SVMs)[23] are a set of supervised learning methods which learn from the dataset and used for classification. Given a set of training examples, each marked as belonging to one of two classes, an SVM algorithm builds a model that predicts whether a new example falls into one class or the other. Simply speaking, we can think of an SVM model as representing the examples as points in space, mapped so that each of the examples of the separate classes are divided by a gap that is as wide as possible. New examples are then mapped into the same space and classified to belong to the class based on which side of the gap they fall on.

More formally it constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can then be used for classification as described above. Now if we view a data point as an n dimensional vector, we want to separate the points by an n-1 dimensional hyperplane.Of the many hyperplanes that might classify the data, One reasonable choice is such that the distance from the hyperplane to the nearest data point on each side is maximized. This is what is called a maximum margin hyperplane.

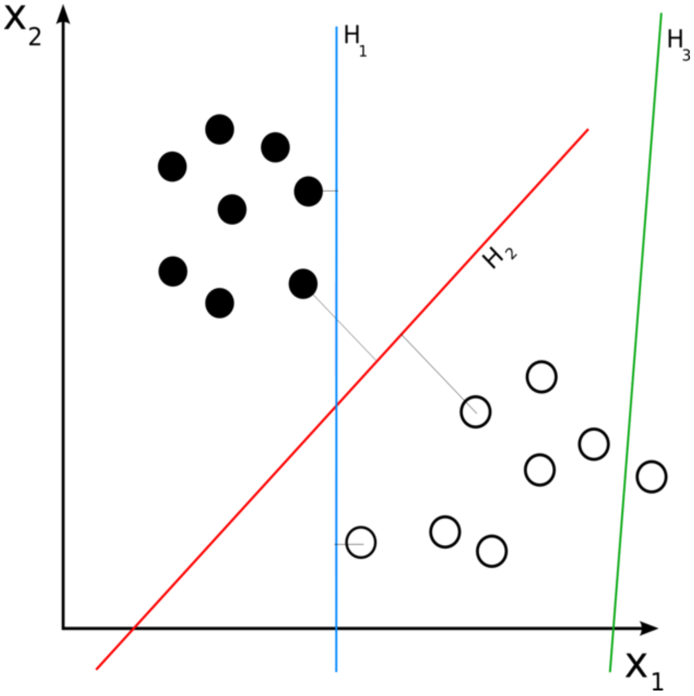


FIGURE 3.9: CLASSIFICATION USING SVM

DECISION TREE CLASSIFIERS

The goal is to create a model that predicts the value of a target variable based on several input variables.. A decision tree[24] or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature.

The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning.

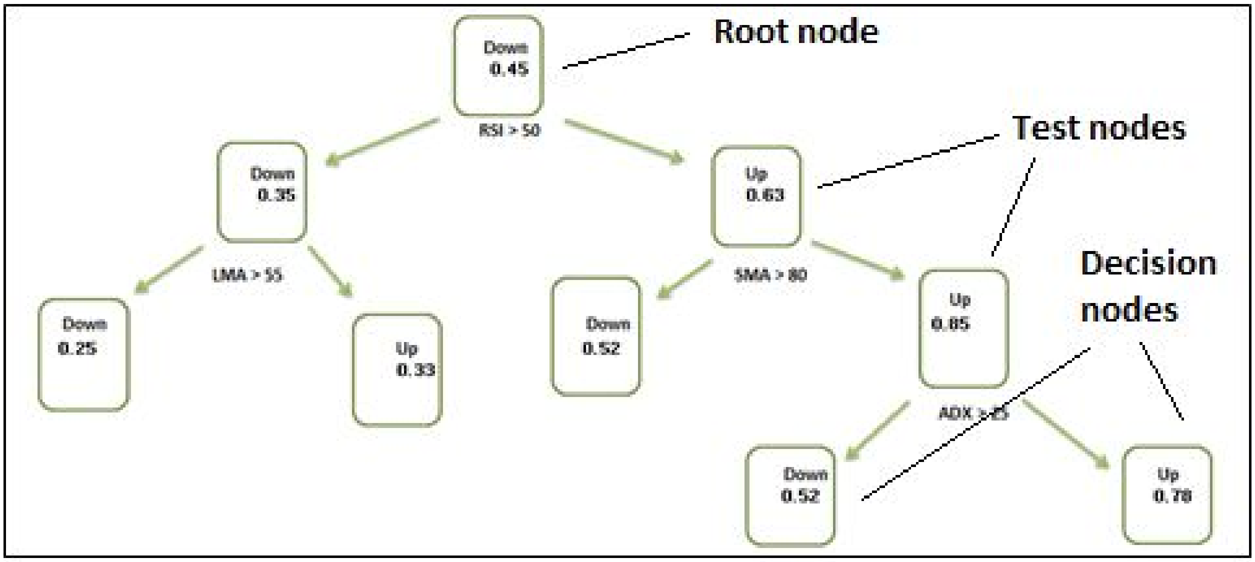


FIGURE 3.10: CLASSIFICATION USING DECISION TREE

A decision tree structure consists of a root node, test nodes, and decision nodes (leaves). The root node is the main node in a decision tree. The final node is the leaf node. In the decision tree diagram, the nodes containing the Up/Down class along with the probability value indicating the probability of the target attribute are the leaf nodes.

Thus, the entire dataset gets classified by navigating from the root node of the tree decision tree down to the leaf, according to the set criterias.

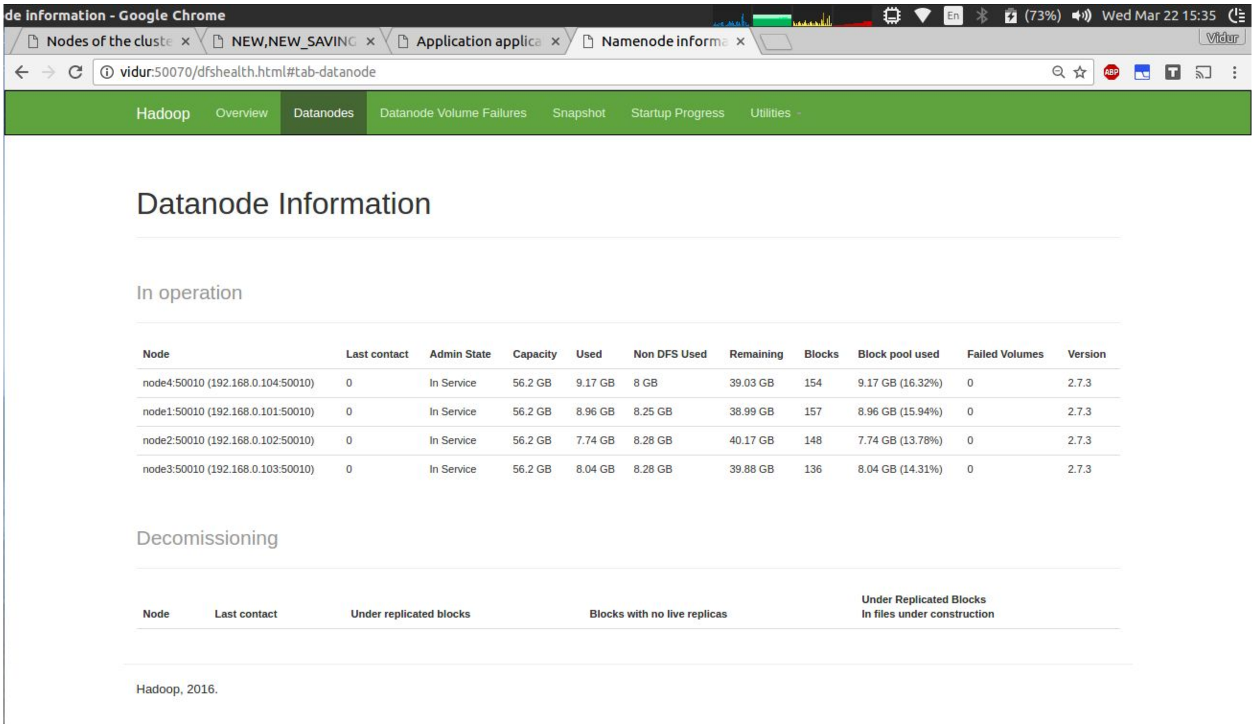
IMPLEMENTATION DETAILS

SETTING UP THE HADOOP CLUSTER

This module describes the setting up of the Hadoop Cluster for this thesis. The cluster setup comprised of 4 Desktop PCs and 1 Laptop along with a router of interconnectivity. It was setup in Graphics Lab (Room No 103/V) at NSIT, New Delhi. We tried to follow the approach mentioned in [link to guide here] with minor variations. Initially, we set up a single node cluster on a Laptop Computer. With assistance from various technology blogs, we were able to scale up to a multi node cluster with 2 Laptops. On this setup we tested a few basic MapReduce jobs. After successfully obtaining some results we moved on to setting the cluster in the Graphics Lab. Several experiments were conducted to evaluate and analyze the HDFS performance and capability to perform MapReduce jobs.

The hardware used in the cluster is given below:-

1. Laptop Computer :- This machine acted as the NameNode in the Cluster. Manufactured by Dell, it has the following specifications - Intel$^{(R)}$ Core$^{(TM)}$ i7-4700MQ CPU, 8GB DDR3 RAM, 128GB Hard Drive.
2. Desktop Computers :- Four machines provided by Computer Engineering Department of NSIT were used which acted as DataNodes in the Cluster. Manufactured by Acer, they have the following specifications - Intel$^{(R)}$ Core$^{(TM)}$ i7-6700K CPU, 4GB DDR3 RAM, 50GB Hard Drive.
3. Router :- A standard home router was used to interconnect the machines. It was a 150Mbps Wireless N Router manufactured by Binatone, model number WR1500N.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
node4:50010 (192.168.0.104:50010)	0	In Service	56.2 GB	9.17 GB	8 GB	39.03 GB	154	9.17 GB (16.32%)	0	2.7.3
node1:50010 (192.168.0.101:50010)	0	In Service	56.2 GB	8.96 GB	8.25 GB	38.99 GB	157	8.96 GB (15.94%)	0	2.7.3
node2:50010 (192.168.0.102:50010)	0	In Service	56.2 GB	7.74 GB	8.28 GB	40.17 GB	148	7.74 GB (13.78%)	0	2.7.3
node3:50010 (192.168.0.103:50010)	0	In Service	56.2 GB	8.04 GB	8.28 GB	39.88 GB	136	8.04 GB (14.31%)	0	2.7.3

Decomissioning

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

Hadoop, 2016.

FIGURE 4.1: HDFS DATANODE INFORMATION

DATA GATHERING AND AGGREGATION

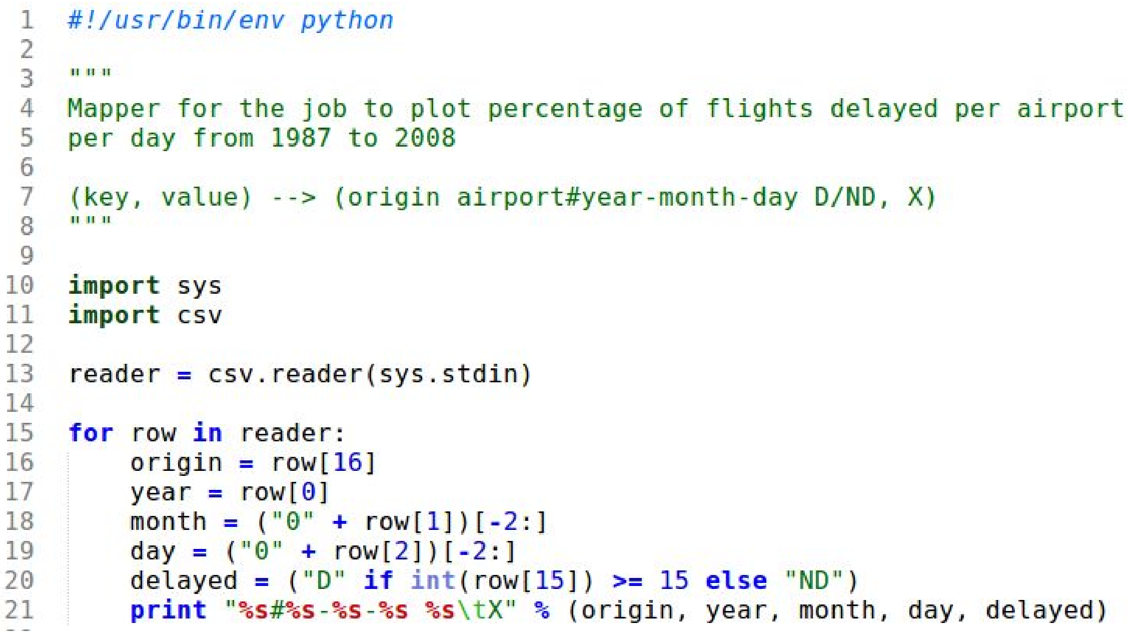
The data for the project was the publicly released dataset called "*Airline on-time performance*" which was used for Data Expo organised by Statistical Computing in 2009. The data consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. This is a large dataset as there are nearly 120 million records in total, and takes up 1.6 gigabytes of space compressed and 12 gigabytes when uncompressed.

Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime
ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsed Time
CRSElapsed Time	AirTime	ArrDelay	DepDelay	Origin	Dest
Distance	TaxiIn	TaxiOut	Cancelled	Cancellation Code	Diverted
CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraft Delay	

TABLE 4.1: COLUMNS IN FLIGHTS DATASET

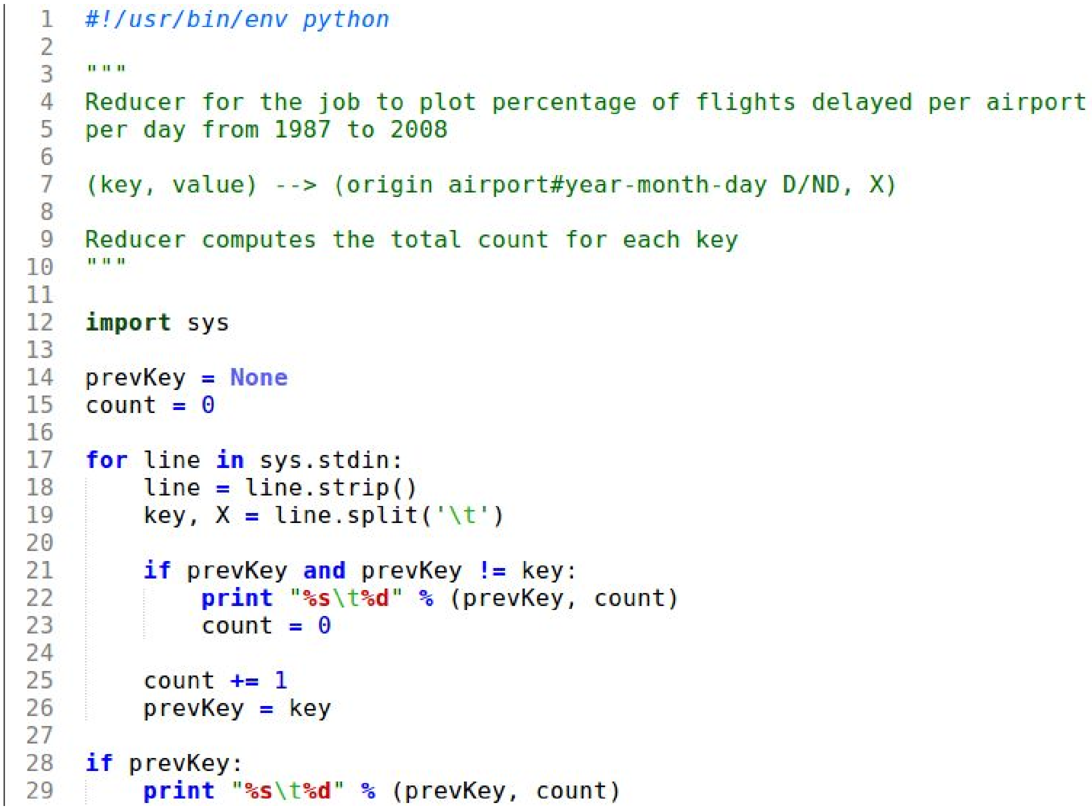
After downloading the data, we uploaded it to the Hadoop Distributed File System (HDFS) which was running on our cluster. We ran various MapReduce jobs to aggregate the data and understand it clearly. The MapReduce jobs, written in Python, were executed on the machines using the Hadoop Streaming utility. It is a generic API which allows writing Mappers and Reduces in any language. Mappers and Reducers receive their input and output on stdin and stdout as (key, value) pairs.

In the figures below is a MapReduce job, written in Python Programming Language, for computing the number of flights which were delayed and not delayed for each day from October 1987 to December 2008 for each airport. It can be observed that both the mapper and reducer read data from stdin and write out to stdout.



```python
#!/usr/bin/env python

"""
Mapper for the job to plot percentage of flights delayed per airport
per day from 1987 to 2008

(key, value) --> (origin airport#year-month-day D/ND, X)
"""

import sys
import csv

reader = csv.reader(sys.stdin)

for row in reader:
    origin = row[16]
    year = row[0]
    month = ("0" + row[1])[-2:]
    day = ("0" + row[2])[-2:]
    delayed = ("D" if int(row[15]) >= 15 else "ND")
    print "%s#%s-%s-%s %s\tX" % (origin, year, month, day, delayed)
```

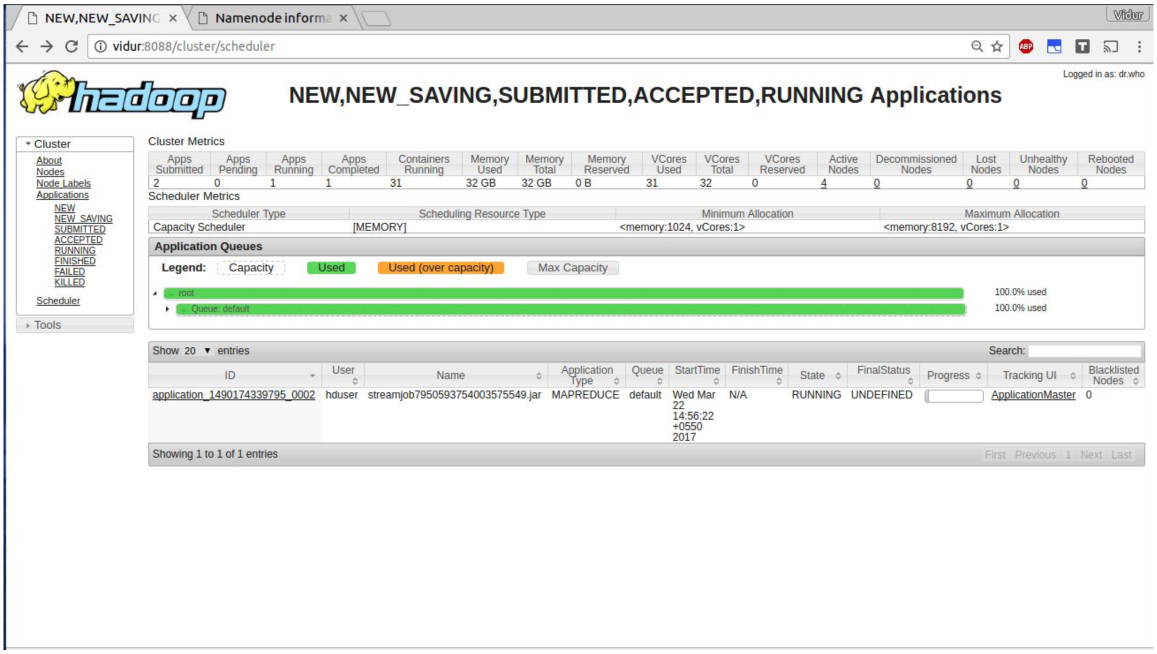
FIGURE 4.2: SAMPLE CODE FOR MAPPER



```python
#!/usr/bin/env python

"""
Reducer for the job to plot percentage of flights delayed per airport
per day from 1987 to 2008

(key, value) --> (origin airport#year-month-day D/ND, X)

Reducer computes the total count for each key
"""

import sys

prevKey = None
count = 0

for line in sys.stdin:
    line = line.strip()
    key, X = line.split('\t')

    if prevKey and prevKey != key:
        print "%s\t%d" % (prevKey, count)
        count = 0

    count += 1
    prevKey = key

if prevKey:
    print "%s\t%d" % (prevKey, count)
```

FIGURE 4.3: SAMPLE CODE FOR REDUCER

We monitored the progress of the MapReduce jobs using the YARN's application tracker UI page. Screenshots for the same are included below. The metrics include Total Memory, Memory Used, Total VCores, VCores Used, No. of active nodes, etc.



NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	1	1	31	32 GB	32 GB	0 B	31	32	0	4	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:1>

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity

root 100.0% used

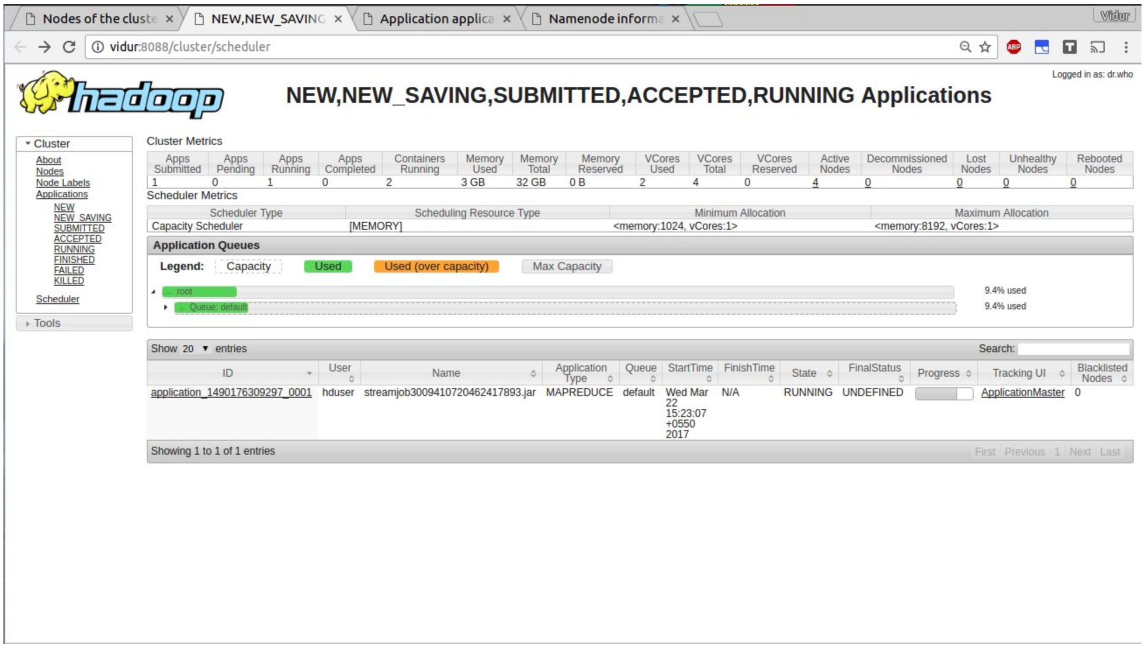
Queue: default 100.0% used

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1490174339795_0002	hduser	streamjob7950593754003575549.jar	MAPREDUCE	default	Wed Mar 22 14:56:22 +0550 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last

FIGURE 4.4: CLUSTER METRICS DURING EXECUTION OF MAP PHASE



NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	1	0	2	3 GB	32 GB	0 B	2	4	0	4	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:1>

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity

root 9.4% used

Queue: default 9.4% used

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1490176309297_0001	hduser	streamjob3009410720462417893.jar	MAPREDUCE	default	Wed Mar 22 15:23:07 +0550 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last

FIGURE 4.5: CLUSTER METRICS DURING EXECUTION OF REDUCE PHASE

DATA EXPLORATION AND VISUALISATION

Choosing a delay threshold

We are planning to develop a binary classifier that predicts whether a flight was delayed or not. Since the number of minutes a flight gets delayed is an analogous function, we need to set a threshold value at which we can claim that the flight was indeed delayed.

Therefore, as a first exploratory analysis, we consider the observed probability of delay in minutes on the entire dataset. We believed that the most effective way to visualise this is through a histogram, looking at departure and arrival delays separately.

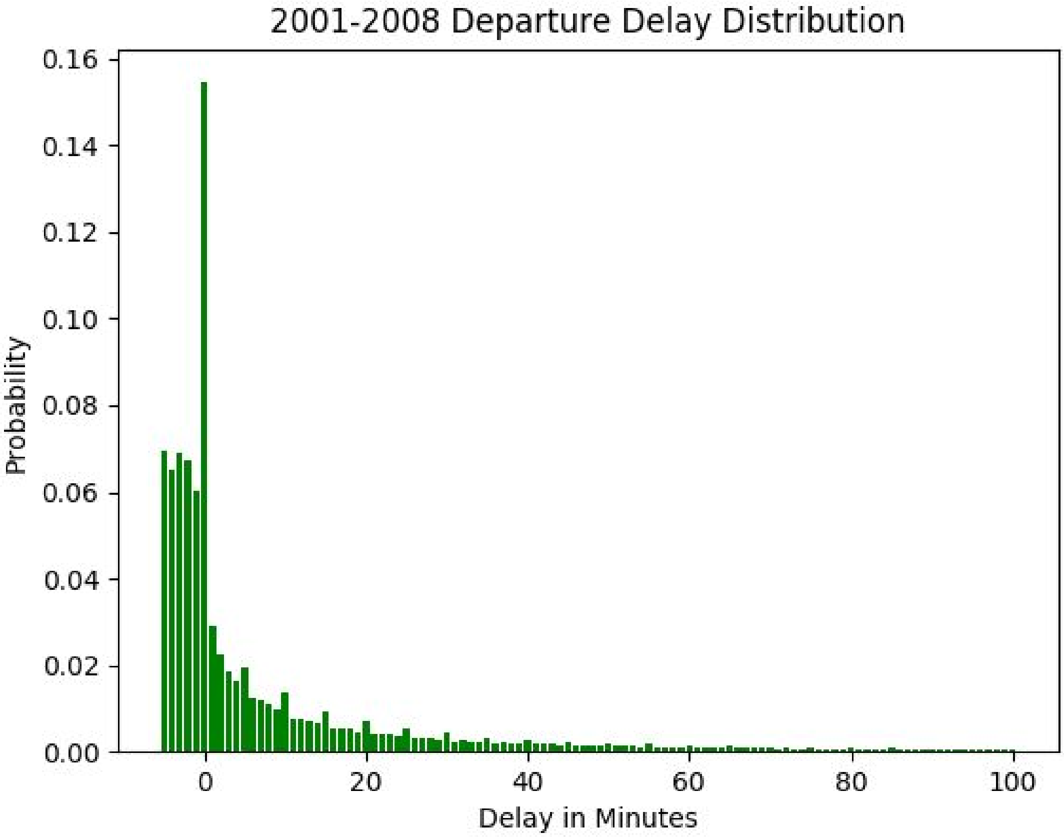


FIGURE 4.6: HISTOGRAM DEPICTING DEPARTURE DELAY DISTRIBUTION

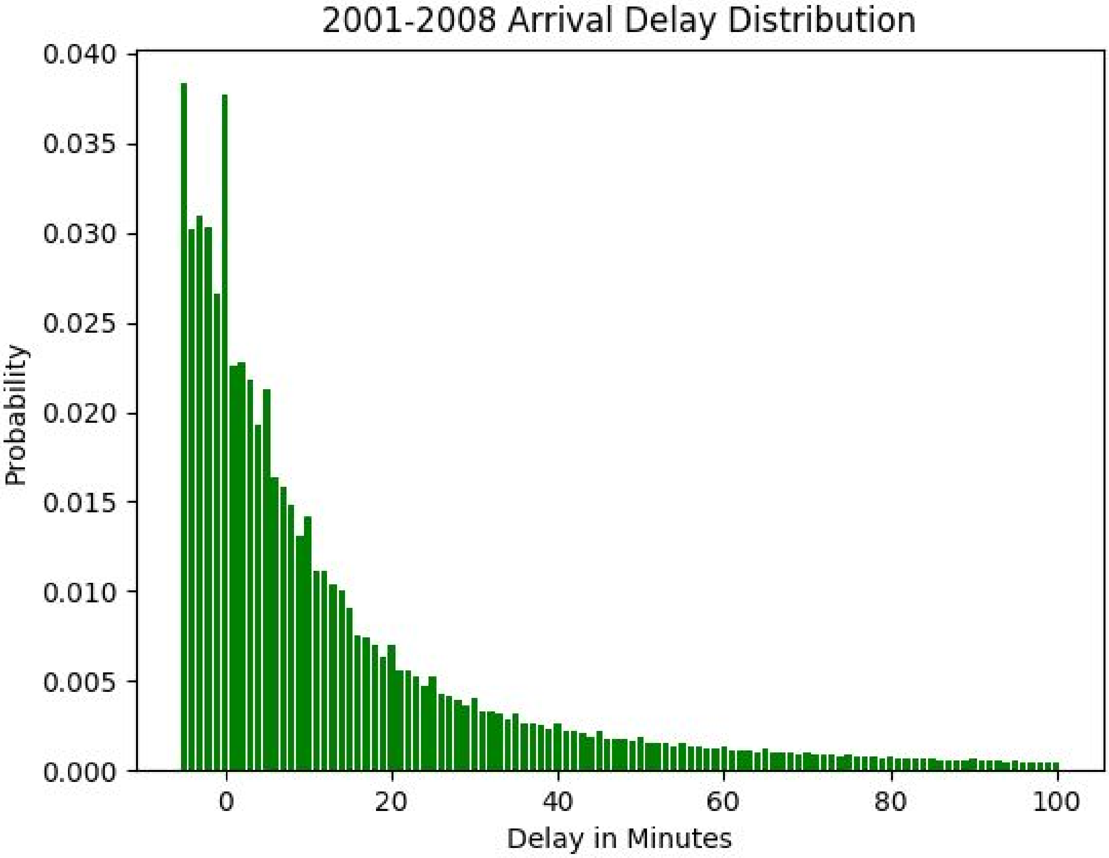


FIGURE 4.7: HISTOGRAM DEPICTING ARRIVAL DELAY DISTRIBUTION

We notice a much higher probability of short delays, even negative in certain cases (which can be considered as advances), for departure delays and a wider distribution (in minutes) for arrivals. We observe from the long right-hand tails that some flights get delayed for very long times, over two hours in some cases. On the other hand, the delays are usually centered around just below zero. In both cases, the mode of the distribution is less than zero, meaning most of the flights leave from gate and arrive at gate even before the published schedule time of departure and arrival.

The x-axis for the two plots are to scale. As a result, we can see that the arrival delay distribution, compared with the departure delay distribution, leans toward left. A flight delay is defined as the time difference between the scheduled time of an event compared to the actual time of the event. Airlines usually put some slack time in a flight to ensure on-time arrival. It is usually done to improve the track record of the airlines in the public's eye.

Therefore, the distribution of the difference in the departure delay and arrival delay indicates that some departure delays are recovered during the flights due to the extra amount of time embedded in the flight time between two airports. In terms of analysis, it makes sense to consider separately departure and arrival delays, since the impacting factors may be different for each.

Furthermore, it can be clearly seen from the graph that there are a bulk of flights with departure delay centered near the zero mark on the x-axis. This value falls greatly near the 15 minute mark. Therefore, we decided to go with 15 minute as the delay threshold in our binary classifier.

Month of flight as a parameter for the model

Now, we consider the impact of month of flight on the delays. We would expect that winter months have the longest delay. A bar chart with average departure and arrival delay in minutes plotted by month is the most effective way to see the potential effect of month.

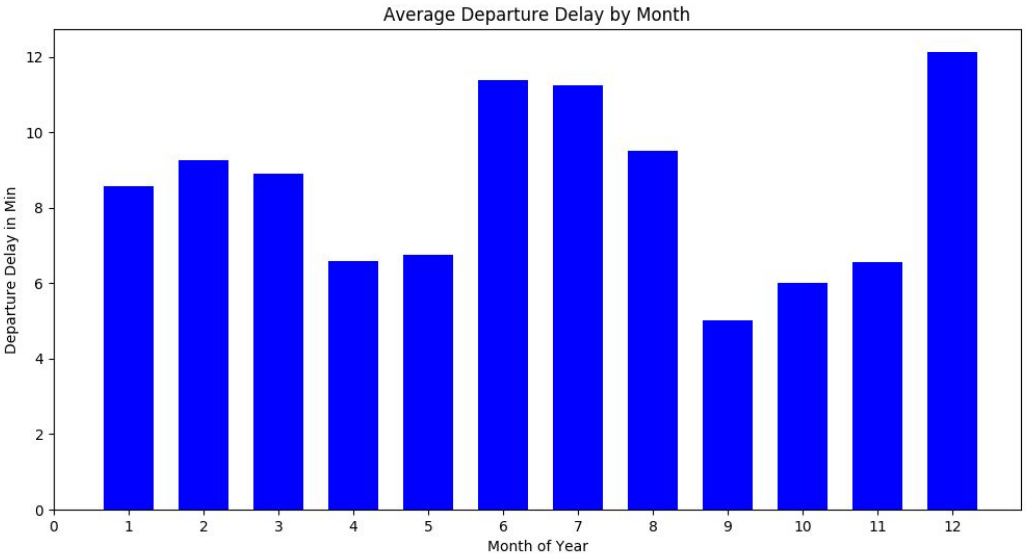


FIGURE 4.8: BAR CHART DEPICTING AVERAGE DEPARTURE DELAY PER MONTH

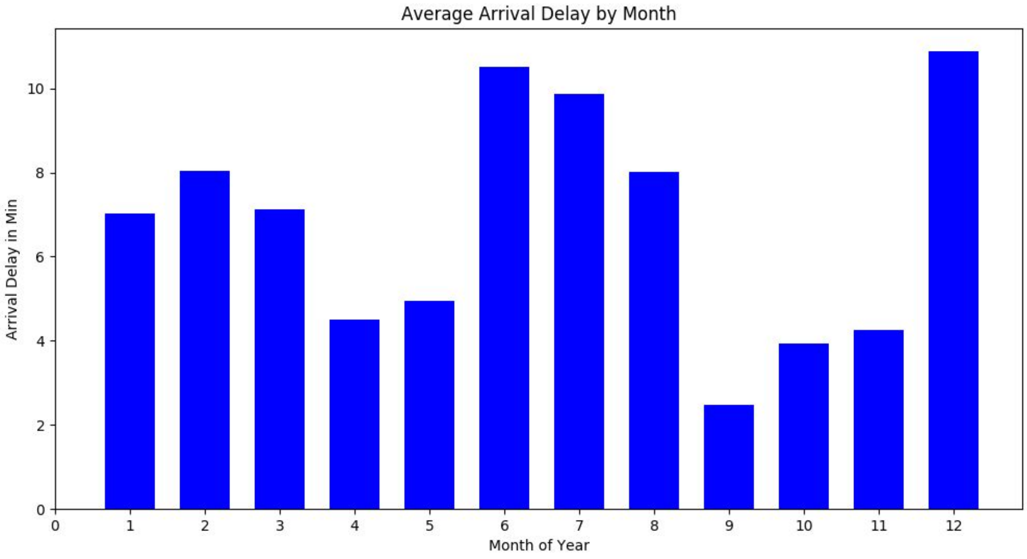


FIGURE 4.9: BAR CHART DEPICTING AVERAGE ARRIVAL DELAY PER MONTH

For both departures and arrivals, the impact of the month of December is clear - the average minutes by which a flight gets delayed is very high in this month which can be used to further infer that the highest delays are in that month. On the other hand, we observe that September, October and November are the months with the least amount of delay. The summer months, June and July, mark the summer holidays in the US and as a result there are more number of flights in these months as compared to the adjacent ones. Therefore, these are also marked by higher delays. Also, we can see that the month of February posts higher delay values as compared to its adjacent months as well.

The reason for winter's high delay values is probably because of snowstorms in the northeast of the US and the celebration of Christmas. Also, in summer, thunderstorms in Dallas Forth Worth (DFW) and Chicago O'Hare International Airport (ORD) can cause high delay impact to the rest of country. While it can be argued that a snow/thunderstorm may only affect operations at one airport or two. We can safely say that delay propagation, which marks as the major contributor for flight delay, can cause ripple effects on delay to downstream flight operations.

Hour of day as a parameter for the model

We think that the time of day should also have an impact on the on-time performance of an airline. Normally, flight delays cumulate throughout the day through propagation, with a knock-on effect of delayed flights provoking other delays because of tight schedules and runway congestion. Again over here we plot the average delay by hour of day in a bar chart for both departure and arrival.

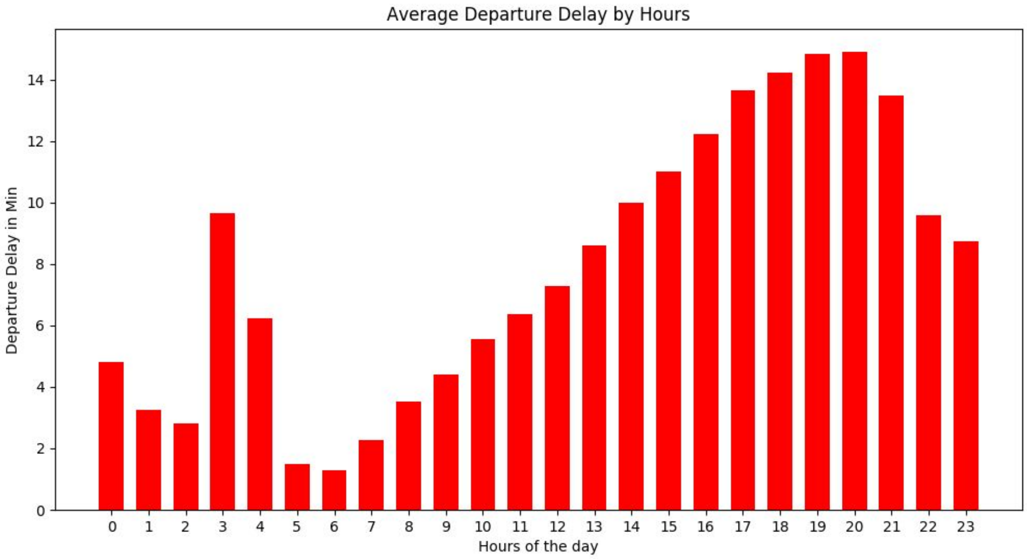


FIGURE 4.10: BAR CHART DEPICTING AVERAGE DEPARTURE DELAY PER HOUR OF DAY

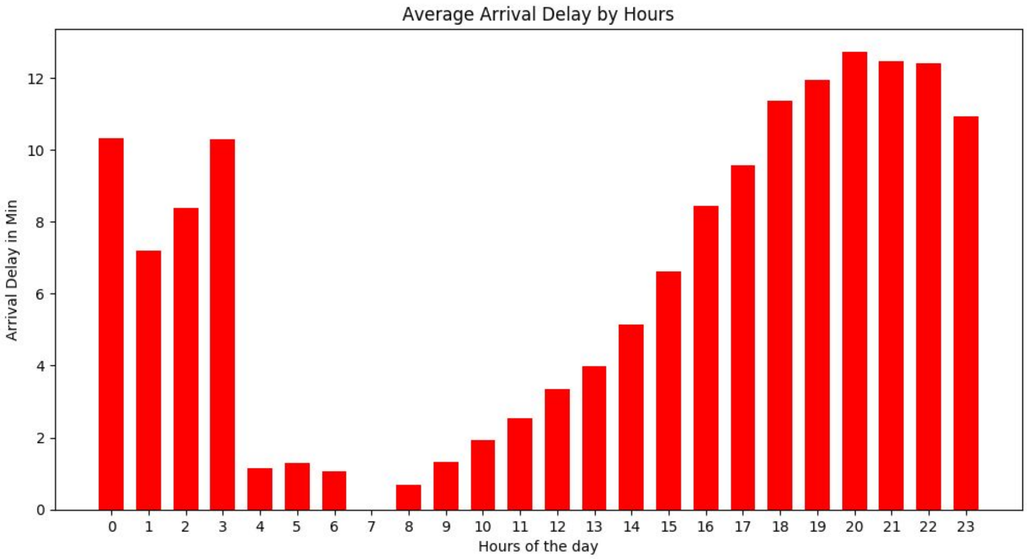


FIGURE 4.11: BAR CHART DEPICTING AVERAGE ARRIVAL DELAY PER HOUR OF DAY

From the plots ,we see a marked "V" shaped decline in delay with the lowest delays in early morning hours. Both departure and arrival delays accumulate from the earlier morning hours reaching their peaks in the evening hours. For departure, the highest mean delay is during prime-time of 18:00 to 21:00, and for arrivals, it is slightly later which peaks at around 22:00. This could be justified by arguing that average flight duration is just a few hours. The increasing of flight delay by the hours of the day is mainly caused by flight delay propagation. Although a flight is built with scheduled slack time for unforeseeable flight delay during the flight operations, it is not sufficient to cover all types of delay. As a result, if a flight is delayed, the next flight has to wait for the late arrival flight to be ready before it can be operated. Hence, flight delays for both departure and arrival flights do increase over time.

As a conclusion from the above exploration, we see that the variation in average delay by hour implies that 'Hour of Day' should be a good predictor of flight delay.

Day of week as a parameter for the model

We think that the day of the week should also have an impact on the on-time performance of an airline. One reason might be the variation in number of flights on different weekdays. Over here we plot the average delay by day of the week in a bar chart for both departure and arrival.

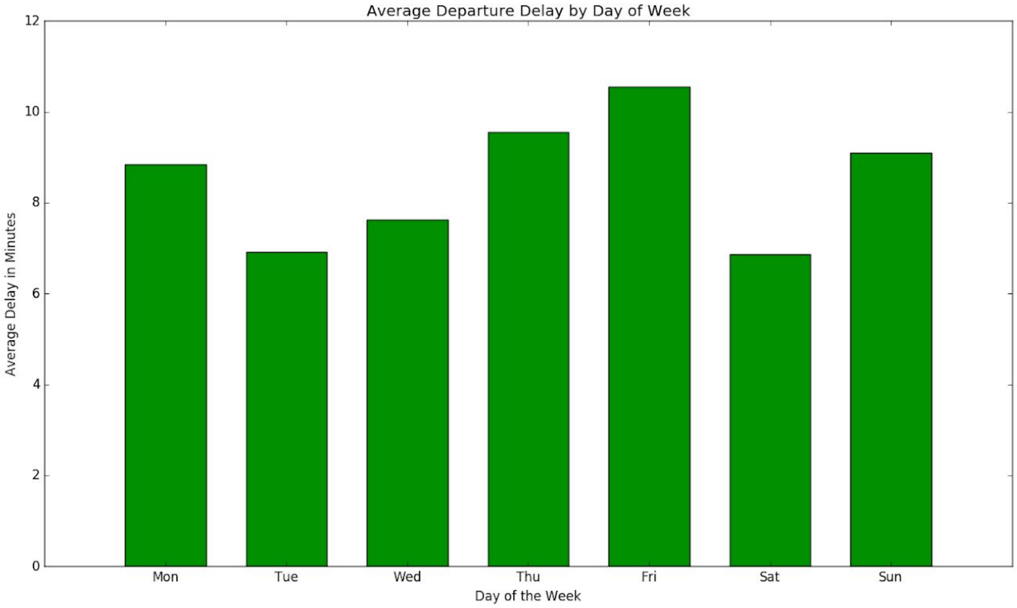


FIGURE 4.12: BAR CHART DEPICTING AVERAGE DEPARTURE DELAY PER DAY OF WEEK

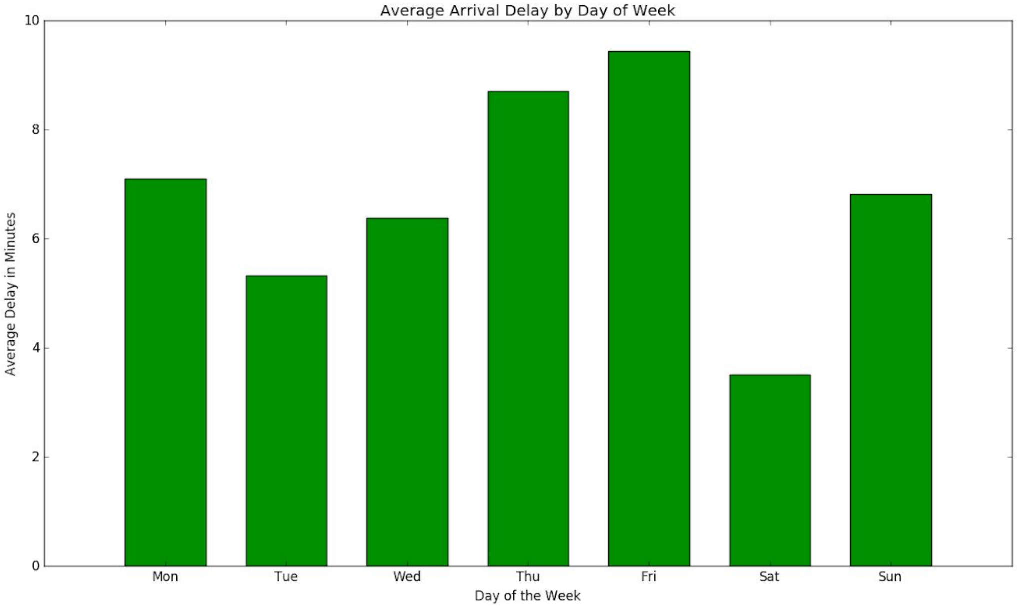


FIGURE 4.13: BAR CHART DEPICTING AVERAGE ARRIVAL DELAY PER DAY OF WEEK

It is evident by the two figures that the delays are quite high on Thursdays and Fridays.

Carrier as a parameter for the model

We think that a carrier should also have an impact on the on-time performance of a flight. Carrier delay is within the control of the airline operator. Again over here we plot the average delay by carrier in a bar chart for both departure and arrival.

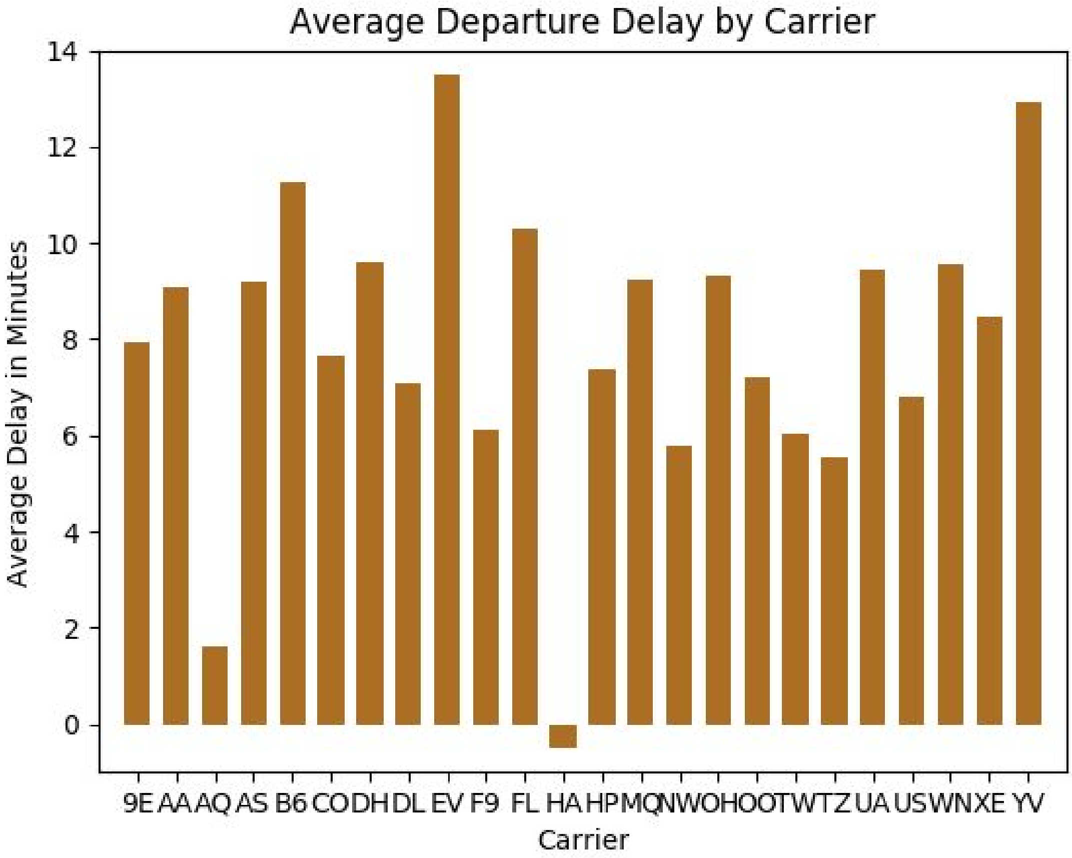


FIGURE 4.14: BAR CHART DEPICTING AVERAGE DEPARTURE DELAY PER CARRIER

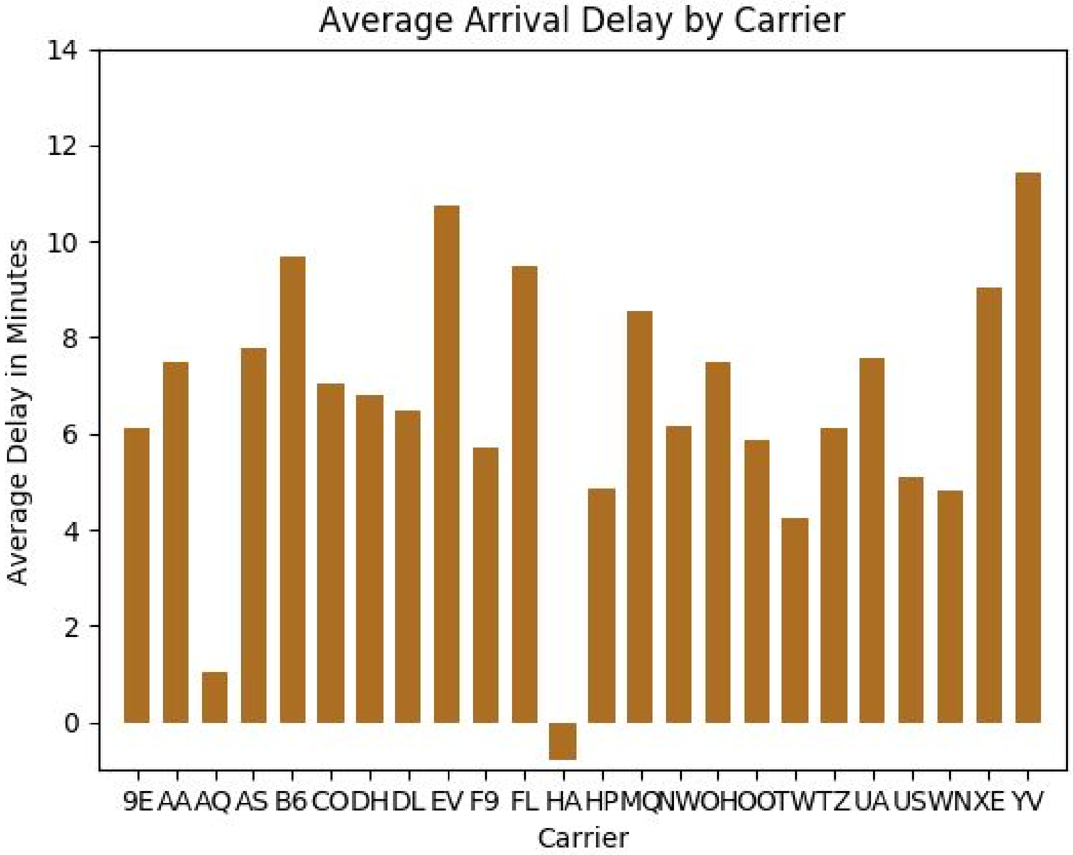


FIGURE 4.15: BAR CHART DEPICTING AVERAGE ARRIVAL DELAY PER CARRIER

Examples of occurrences that may determine carrier delay are: aircraft cleaning, aircraft damage, awaiting the arrival of connecting passengers or crew, baggage, bird strike, cargo loading, catering, computer, outage-carrier equipment, crew legality (pilot or attendant rest), damage by hazardous goods, engineering inspection, fueling, handling disabled passengers, late crew, lavatory servicing, maintenance, oversales, potable water servicing, removal of unruly passenger, slow boarding or seating, stowing carry-on baggage, weight and balance delays. As different airlines have different set of policies to deal with these scenarios, we believe that carrier can also affect the average delay minutes.

From the data it can be clearly seen that the average flight delays vary considerably. The analysis is also affected by the number of flights for each carrier. Some carriers with small number of flights. Like Aloha Airlines (AQ) or Hawaiian Airlines (HA), have lowest average delays. Interestingly for HA the average delay is in fact negative. Both AQ and HA mostly operate flight between Hawaii and Continental United States. Since the flights are mostly 5-6 hours, these flights are usually exempt from Ground Delay Programs (GDPs) when an airport encounter inclement weather conditions. This means that no delay is imposed by air traffic management system. Therefore, the average delays for these two carriers can be quite low.

Number of days from nearest holiday as a parameter for the model

We believe that near the holidays there might be more number of flights that get delayed. For this we used only the federal holidays identified by the United States Congress. Unlike many other countries, there are no ‘national holidays’ in the United States because Congress only has constitutional authority to create holidays for federal institutions. Most federal holidays are also observed as state holidays. Some of these holidays include - Independence Day, Memorial Day, Labor Day, etc. An interesting observation can be made over here is that these holidays are not always observed on the same day. When a federal holiday falls on a Saturday, it is usually observed on the preceding Friday. When the holiday falls on a Sunday, it is usually observed on the following Monday. Here we are plotting the percentage of flights delayed against the days in the year. The holidays are marked by dotted lines parallel to the Y-axis.

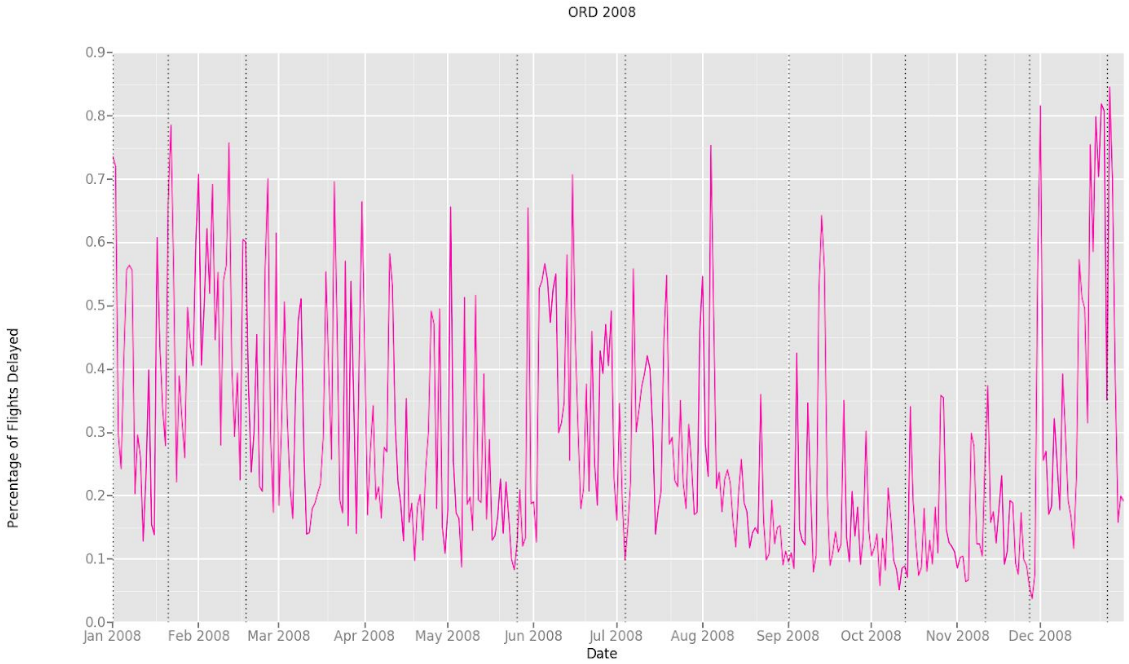


FIGURE 4.16: LINE CHART DEPICTING PERCENTAGE OF FLIGHTS DELAYED PER DAY FOR ORD IN 2008

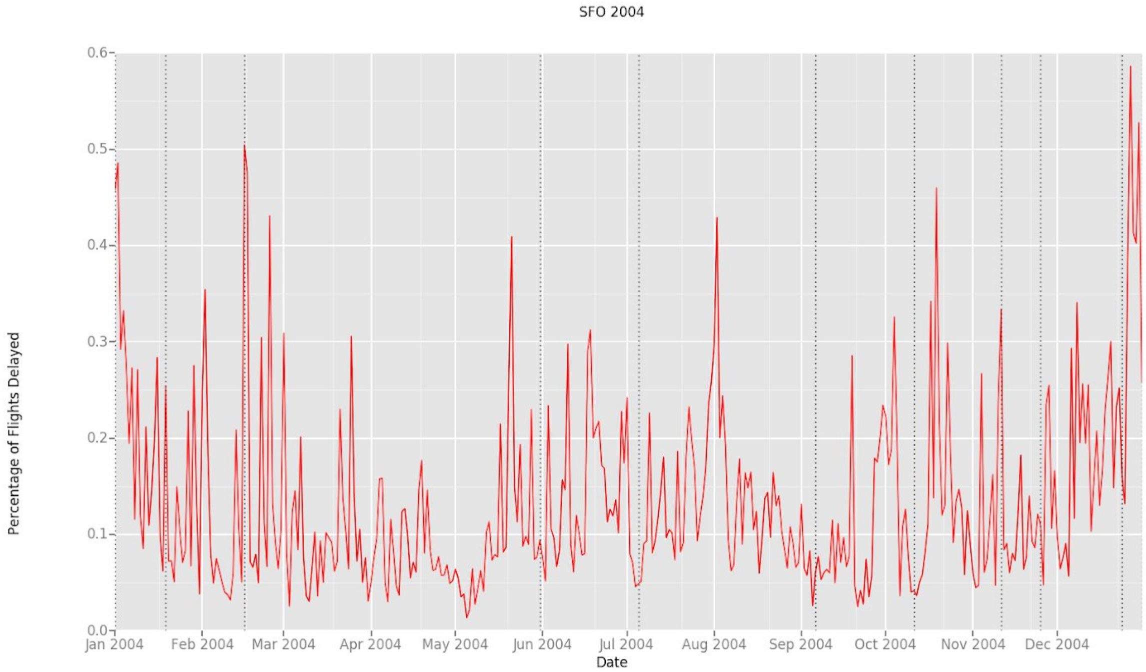


FIGURE 4.17: LINE CHART DEPICTING PERCENTAGE OF FLIGHTS DELAYED PER DAY FOR SFO IN 2004

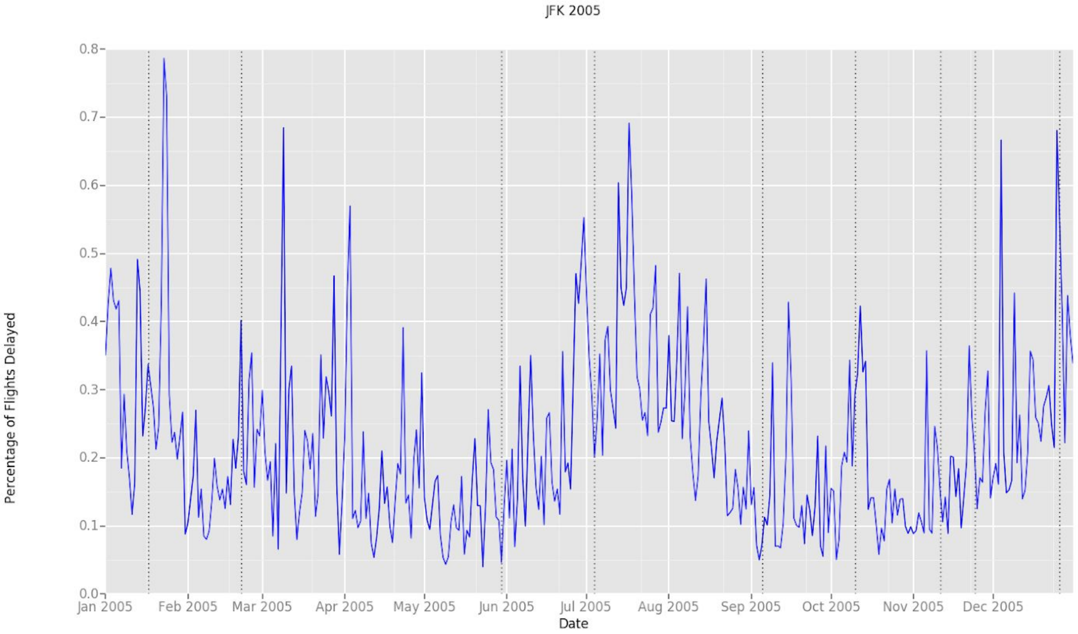


FIGURE 4.18: LINE CHART DEPICTING PERCENTAGE OF FLIGHTS DELAYED PER DAY FOR JFK IN 2005

The plots for airports ORD in Chicago, SFO in San Francisco and JFK in New York are shown for the years 2008, 2004 and 2005. Based on these graphs we can infer that there are more number of flights getting delayed in the vicinity of the holidays falling in the months of January, February, November and December. Therefore, adding this parameter could surely help our prediction model.

SETTING UP APACHE SPARK

For the next part of this thesis to implement the prediction model on the cluster, Apache Spark was used. Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. We used the MLlib(Spark's machine learning library) to build and evaluate our supervised learning model. We configured spark with YARN cluster manager which uses HDFS as underlying storage. Spark setup was done as per the instructions given in Apache Spark documentation[16]. spark-submit script in Spark's bin directory is used to launch applications on the cluster. It uses YARN cluster manager[17] through a uniform interface so we didn't have to configure our application specially for each one. In order to distribute the code to a spark cluster the code was bundled along with its dependencies. sbt[18] extension for Eclipse was used to complete this task. Once an application is bundled, it can be launched using the bin/spark-submit script. This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports:

```
./bin/spark-submit  --class <main-class>  --master <master-url>  --deploy-mode <deploy-mode>  --conf
<key>=<value>  ... <application-jar>  [application-arguments]
```

Some of the commonly used options are:

- --class: The entry point for your application (e.g. org.apache.spark.examples.SparkPi)
- --master: The master URL for the cluster (e.g. spark://23.195.26.187:7077)
- --deploy-mode: Whether to deploy your driver on the worker nodes (cluster) or locally as an external client (default: client)

- --conf: Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap “key=value” in quotes (as shown).
- application-jar: Path to a bundled jar including your application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an hdfs:// path or a file:// path that is present on all nodes.
- application-arguments: Arguments passed to the main method of the main class, if any

TRAINING THE PREDICTION MODEL

Apache Spark's basic data abstraction is that of an RDD (resilient distributed dataset), which is a fault-tolerant collection of elements that can be operated on in parallel across your Hadoop cluster.

Spark's API for Scala supports a variety of transformations such as map() and flatMap(), filter(), join(), and others to create and manipulate RDDs.

Features for the model:

- month
- day of month
- day of week
- hour of the day
- Carrier
- Destination airport
- Distance
- Days from Nearest Holiday

Preprocessing the input

Spark RDDs were used to perform the pre-processing, transforming the raw flight delay dataset into the two feature matrices, namely the training set and the test set. We gathered a list of federal holidays observed in the United States from the Legal Information Institute, Cornell Law School's website[21].

Some steps involved in preprocessing include:-

1. Reading the raw input file with Spark's SparkContext.textFile method which results in an RDD.
2. Parsing each row with CSVReader into fields, and populate it into a class that encapsulates a flight delay record object. Here the days_from_nearest_holidays field is computed.
3. A sequence of RDD transformations on the input RDD were performed to make sure we only have rows that correspond to flights that did not get cancelled.

With the training and the testing datasets represented as RDDs, we now build a predictive model using Spark's MLlib machine learning library.

MLlib is Spark’s scalable machine learning library, which includes various learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and others.

To use MLlib's machine learning algorithms, we first parsed our feature matrices into RDDs of LabeledPoint objects. LabeledPoint is MLlib's abstraction for a feature vector accompanied by a label. We consider flight delays of 15 minutes or more as "delays" and mark it with a label of 1.0, and under 15 minutes as "non-delay" and mark it with a label of 0.0. We also used MLlib's StandardScaler class to normalize our feature values for both training and validation sets.

We used the RDD cache method to ensure that the computed RDDs, namely parsedTrainData, scaledTrainData, parsedTestData and scaledTestData, are cached in memory by Spark and not re-computed with each iteration of the algorithm.

A helper function was also defined to evaluate our metrics: precision, recall, accuracy and the F1-measure[25].

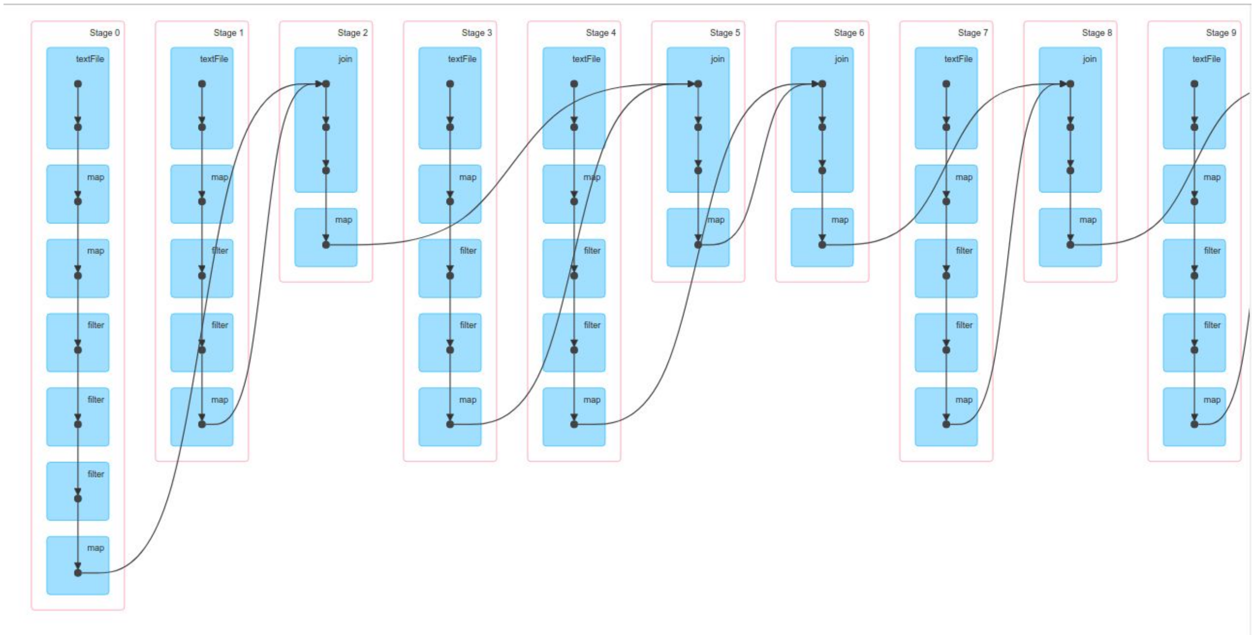
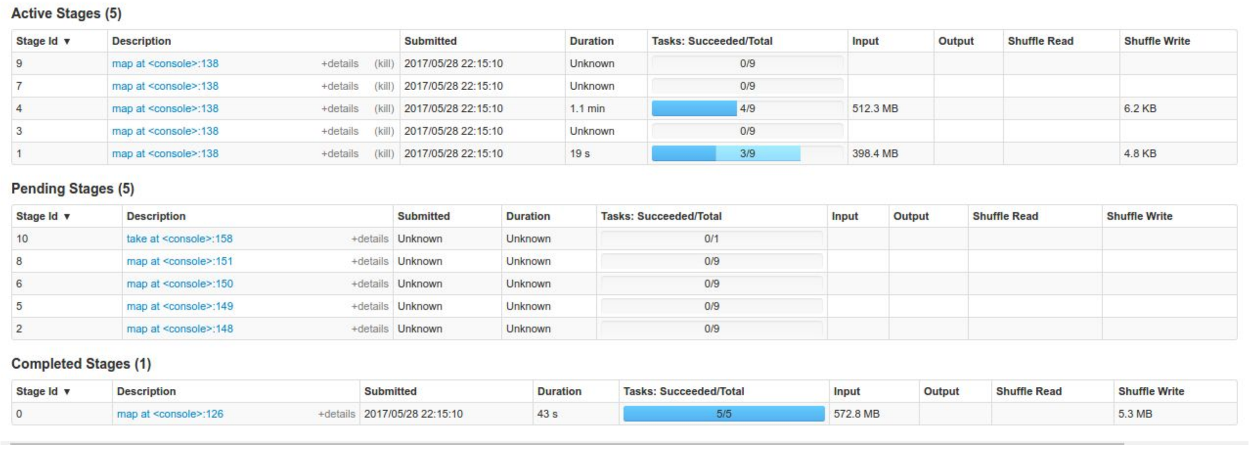


FIGURE 4.19: SPARK DAG VISUALISATION



Active Stages (5)

Stage Id ▾	Description			Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	map at <console>:138	+details	(kill)	2017/05/28 22:15:10	Unknown	0/9				
7	map at <console>:138	+details	(kill)	2017/05/28 22:15:10	Unknown	0/9				
4	map at <console>:138	+details	(kill)	2017/05/28 22:15:10	1.1 min	4/9	512.3 MB			6.2 KB
3	map at <console>:138	+details	(kill)	2017/05/28 22:15:10	Unknown	0/9				
1	map at <console>:138	+details	(kill)	2017/05/28 22:15:10	19 s	3/9	398.4 MB			4.8 KB

Pending Stages (5)

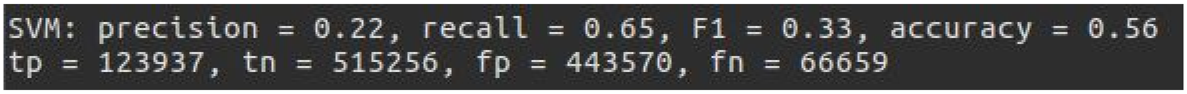
Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
10	take at <console>:158	+details	Unknown	Unknown	0/1				
8	map at <console>:151	+details	Unknown	Unknown	0/9				
6	map at <console>:150	+details	Unknown	Unknown	0/9				
5	map at <console>:149	+details	Unknown	Unknown	0/9				
2	map at <console>:148	+details	Unknown	Unknown	0/9				

Completed Stages (1)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	map at <console>:126	+details	2017/05/28 22:15:10	43 s	5/5	572.8 MB			5.3 MB

FIGURE 4.20: METRICS FOR A SPARK JOB

We built a model using Support Vector Machine with 100 iterations and regularisation parameter as 1, and then used it to predict flight delays over the testing set to measure our performance: precision, recall, F1 and accuracy.



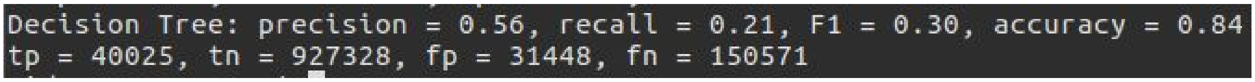
```
SVM: precision = 0.22, recall = 0.65, F1 = 0.33, accuracy = 0.56
tp = 123937, tn = 515256, fp = 443570, fn = 66659
```

FIGURE 4.21: PERFORMANCE METRICS FOR SVM

	Predicted Delay	Predicted Not Delay
Actual Delay	123937	66659
Actual Not Delay	443570	515256

TABLE 4.2: CONFUSION MATRIX FOR SVM

MLlib also has a strong Decision Tree implementation. We built a model using it with max depth 100, max bins 100, "gini" impurity and 2 classes. Again with this model, Flight delays were predicted over the testing set and performance (precision, recall, F1 and accuracy) was measured.



```
Decision Tree: precision = 0.56, recall = 0.21, F1 = 0.30, accuracy = 0.84
tp = 40025, tn = 927328, fp = 31448, fn = 150571
```

FIGURE 4.22: PERFORMANCE METRICS FOR DECISION TREE

	Predicted Delay	Predicted Not Delay
Actual Delay	40025	150571
Actual Not Delay	31448	927328

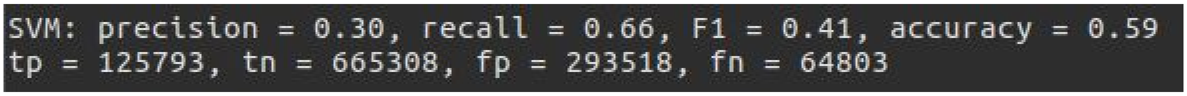
TABLE 4.3: CONFUSION MATRIX FOR DECISION TREE

Including weather data along with the flights dataset

We now enrich the dataset by integrating weather data[26] into our feature matrix, thus hoping to achieve better predictive performance overall for our model. To accomplish this with Apache Spark, we rewrite our previous preprocessing function to extract the same base features from the flight delay dataset, and also join those with five variables from the weather datasets:

- Minimum temperature (tenths of degrees C) - TMIN
- Maximum temperature (tenths of degrees C) - TMAX
- Precipitation (tenths of mm) - PRCP
- Snowfall (mm) - SNOW
- Average daily wind speed (tenths of meters per second) - AWND

We repeated the previous models of SVM and Decision Tree with our enriched feature set. As before, we created an RDD of LabeledPoint objects, and normalized our dataset with MLlib's StandardScaler.

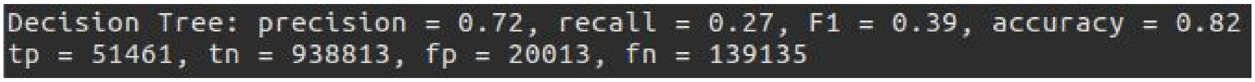


```
SVM: precision = 0.30, recall = 0.66, F1 = 0.41, accuracy = 0.59
tp = 125793, tn = 665308, fp = 293518, fn = 64803
```

FIGURE 4.23: PERFORMANCE METRICS FOR SVM (WITH WEATHER DATA)

	Predicted Delay	Predicted Not Delay
Actual Delay	125793	64803
Actual Not Delay	293518	665308

TABLE 4.4: CONFUSION MATRIX FOR SVM (WITH WEATHER DATA)



```
Decision Tree: precision = 0.72, recall = 0.27, F1 = 0.39, accuracy = 0.82
tp = 51461, tn = 938813, fp = 20013, fn = 139135
```

FIGURE 4.24: PERFORMANCE METRICS FOR DECISION TREE (WITH WEATHER DATA)

	Predicted Delay	Predicted Not Delay
Actual Delay	51461	139135
Actual Not Delay	20013	938813

TABLE 4.5: CONFUSION MATRIX FOR DECISION TREE (WITH WEATHER DATA)

As expected, the improved feature set increased the accuracy of our model for both SVM and Decision Tree models.

CONCLUSION

The main goal of this thesis was to analyse big data using the Hadoop framework with focus on building a machine learning model which predicts possibility of delays for airlines. In this thesis, we used both flight data and weather data to predict flight departure delay. Instead of finding the amount of delay, we implemented a binary classifier to categorize if a flight is on-time or delayed (departure delay more than 15 minutes). The accuracy for Decision Tree was found to be better than that of SVM in both the cases. SVM, on the other hand, had a better F1 score. At the end we correctly predicted 97.9% of non-delayed flights with Decision Tree model and 69.38% with SVM model.

We also discussed that that the flight arrival/departure delay is found to show seasonal and weekly patterns, which is related to the schedule performance. The patterns of delay from the flight level in which delays occur are analyzed, and the significant reasons of delay were given out. We argued that the carrier, departure time, season, weekday, arrival time and month also contribute to delay of flights. As proven by the performance of the models with weather data, it can be argued that the weather conditions such as precipitation and wind speed also contribute to flight delay.

FUTURE WORK

Increased numbers of smaller jets, which operate in the same flight regime as the larger jets, means more aircraft competing for the same airspace, thereby increasing congestion and delays. The airline information and aircraft model are not considered in the thesis make the drawbacks. We can gather this data from relevant resources, to further enrich our model.

It is theorized that larger airlines should have increased exposure to delays from weather when they serve more destinations. More destinations mean more potential delays to be spread throughout the system. Because they have more flights, larger airlines can more accurately predict the likelihood of crew sickness or mechanical failure.

We can also use some additional features to further enrich our model. As an example, in the dataset there is a field “tail number”. A tail number uniquely identifies an aircraft. This field can be used to determine the age of the aircraft and this age can be taken into account to check whether older planes suffer more delay or not. We can also use this tail number to trace whether an inbound flight was delayed or not. If it was delayed, this can lead to a cascading effect and can be a major contributor in delay of outbound flight.

Given the amount of data that is available, we can further create sub-models for specific airlines at an airport provided that the airlines have fair share of the operations at an airport.

In addition to this, we can also create sub-models for specific airports taking in account some features that belong to them.

REFERENCES

[1]Tom White. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 3rd edition, 2009.
[2]Laney, Doug (2001-02-06). "3D Data Management: Controlling Volume, Velocity, and Variety". *Meta Group*.
[3]https://github.com/caesar0301/awesome-public-datasets
[4]http://stat-computing.org/dataexpo/2009/
[5]Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *OSDI 2004*
[6]Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler, "The Hadoop Distributed File System"
[7]Matei Zaharia, Mosharaf Chowdhury, et al., "Spark: Cluster Computing with Working Sets"
[8]https://www.python.org/
[9]https://www.scala-lang.org/
[10]https://matplotlib.org/
[11]http://ggplot.yhathq.com/
[12]http://www.numpy.org/
[13]http://pandas.pydata.org/
[14]Vinod Kumar Vavilapalli, et al., "Apache Hadoop YARN: Yet Another Resource Negotiator"
[15]https://spark.apache.org/mllib/
[16]https://spark.apache.org/docs/2.1.0/
[17]https://spark.apache.org/docs/latest/cluster-overview.html#cluster-manager-types
[18]https://github.com/sbt/sbt-assembly
[19]Matei Zaharia, et al. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"
[21]https://www.law.cornell.edu/uscode/text/5/6103#fn002099-ref
[22]Murthy, Arun (2012-08-15). "Apache Hadoop YARN – Concepts and Applications". *hortonworks.com*.
[23]Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, "A Practical Guide to Support Vector Classification"
[24]Zied Elouedi, Khaled Mellouli, Philippe Smets, "Decision trees using the belief function theory"
[25]"EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION", Journal of Machine Learning Technologies
[26]ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/
[27]https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp
[28]Everett B. Peterson, et al., "The Economic Cost of Airline Flight Delay", *Journal of Transportation Economics and Policy, Jan 2013*