HEALTH CARE DATA ANALYTICS USING HADOOP

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Deepthi Yaramala
Spring 2016
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Deepthi Yaramala:

Health Care Data Analytics Using Hadoop

______________________________
Carl Eckberg, Chair
Department of Computer Science

______________________________
Alan Riggins
Department of Computer Science

______________________________
J. Carmelo Interlando
Department of Mathematics and Statistics

April 9, 2016
Approval Date
DEDICATION

I want to dedicate my work to my husband for his immense support and to my parents for their unconditional love. I also dedicate this to my advisor Dr. Carl Eckberg for his guidance and support.
ABSTRACT OF THE THESIS

Health Care Data Analytics Using Hadoop
by
Deepthi Yaramala
Master of Science in Computer Science
San Diego State University, 2016

In today’s world, due to technological advancements, the amount of data that is getting generated is growing rapidly. Enterprises worldwide will need to perform data analytics with these huge data datasets to make business decisions and stay competitive.

Storage of data sets and performing data analytics was traditionally accomplished using RDBMS (Relational Database Management System). However, RDBMS would be inefficient and time consuming when performing data analytics on huge data sets. Hadoop came into existence recently and overcomes the limitations of existing RDBMS by providing simplified tools for efficient data storage and faster processing times for data analytics.

The purpose of this work is to study different Hadoop functionalities in detail and perform data analytics on a health care data set using Hadoop. A health care data set comprising of 1.5 million patient records is considered for the data analysis. Different use cases have been considered and analytics have been performed using MapReduce, Hive and Pig functionalities of Hadoop.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
</tr>
</tbody>
</table>

## CHAPTER

1 INTRODUCTION .................................................................1

1.1 Overview ............................................................................1

1.2 Thesis Contribution..........................................................3

1.3 Thesis Organization ...........................................................3

2 BIG DATA AND HADOOP .........................................................4

2.1 Big Data ..............................................................................4

  2.1.1 Evolution of Data/Big Data ..............................................4

  2.1.2 Characteristics of Big Data ..............................................4

    2.1.2.1 Volume ....................................................................5

    2.1.2.2 Variety ....................................................................5

    2.1.2.3 Velocity ....................................................................6

2.2 Hadoop ..............................................................................6

  2.2.1 Hadoop Architecture ..................................................6

  2.2.2 Hadoop Components ..................................................7

    2.2.2.1 HDFS (Hadoop Distributed File System) .....................7

    2.2.2.2 MapReduce .........................................................7

  2.2.3 Hadoop Characteristics ...............................................9

  2.2.4 Hive ............................................................................9

    2.2.4.1 Hive Architecture ................................................10
2.2.4.2 Hive Components

2.2.5 Pig

2.2.5.1 Pig Architecture

2.2.5.2 Pig Latin Compiler

2.2.5.3 Benefits of Pig

2.2.5.4 Disadvantages of Pig

3 HADOOP INSTALLATION

3.1 Virtual Machine Installation

3.2 Loading Data into HDFS

4 HEALTH CARE DATA ANALYTICS

4.1 Health Care Data Set

4.1.1 Data Set Generation

4.2 Health Care Data Analytics

4.2.1 Patients in Each Blood Group Across Each State

4.2.2 Patients Needing Blood in Each State for Each Year

4.2.3 Disease Affecting the Most for Each Race

4.2.4 Patients Who Had Moderate Side Effects Segregated Across Each Disease

4.2.5 Partitioning Whole Data Set Based on States

4.2.6 Percentages of Insurance Company Usage Per Year for a Given State

4.2.7 Company That Supplied Most Medications in Each Year

4.2.8 Patients Who Had Moderate Side Effects Segregated Across Each State [Implemented Using Hive]

4.2.9 Patients in Each Blood Group Across Each State [Implemented Using Hive]

4.2.10 Patients in Each Blood Group [Implemented Using Pig]

4.2.11 Patients Needing Blood in Each State [Implemented Using Pig]

BIBLIOGRAPHY
LIST OF TABLES

Table 4.1. Health Care Data Set Fields.................................................................24
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Data Analytics using Traditional Systems.</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Characteristics of Big Data.</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Hadoop Architecture.</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Data processing using MapReduce framework.</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Hive Architecture block diagram.</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>Pig Architecture block diagram.</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Virtual Machine installation.</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>Virtual Machine settings.</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>Starting Virtual Machine.</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Finding IP address of VM.</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>WinSCP login screenshot.</td>
<td>19</td>
</tr>
<tr>
<td>3.6</td>
<td>Data sanity check in VM before moving to HDFS.</td>
<td>20</td>
</tr>
<tr>
<td>3.7</td>
<td>Put data into HDFS.</td>
<td>21</td>
</tr>
<tr>
<td>3.8</td>
<td>Data sanity check in HDFS.</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Custom list creation for blood group data.</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Custom distribution illustration for blood group data.</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>In built formula for “Insurance_Claimed” field generation</td>
<td>25</td>
</tr>
<tr>
<td>4.4</td>
<td>Regular Expression used for “Side_Effects” field generation.</td>
<td>25</td>
</tr>
<tr>
<td>4.5</td>
<td>Sample final health care data set in JSON format.</td>
<td>26</td>
</tr>
<tr>
<td>4.6</td>
<td>Converting to JAR file.</td>
<td>28</td>
</tr>
<tr>
<td>4.7</td>
<td>Execution output for use case [Patients in each blood group across each state]</td>
<td>29</td>
</tr>
<tr>
<td>4.8</td>
<td>Output1 for use case [Patients in each blood group across each state]</td>
<td>30</td>
</tr>
<tr>
<td>4.9</td>
<td>Output2 for use case [Patients in each blood group across each state]</td>
<td>30</td>
</tr>
<tr>
<td>4.10</td>
<td>Execution output1 for use case [Patients needing blood in each state for each year]</td>
<td>31</td>
</tr>
</tbody>
</table>
Figure 4.11. Execution output2 for use case [Patients needing blood in each state for each year] ........................................................................................................................................................................32
Figure 4.12. Output1 for use case [Patients needing blood in each state for each year] ........................................33
Figure 4.13. Output2 for use case [Patients needing blood in each state for each year] ........................................33
Figure 4.14. Execution output1 for use case [Disease affecting the most for each race] ........................................................................................................................................................................................................34
Figure 4.15. Execution output2 for use case [Disease affecting the most for each race] ........................................................................................................................................................................................................35
Figure 4.16. Output result for use case [Disease affecting the most for each race] ........................................36
Figure 4.17. Execution output1 for use case [Patients who had moderate side effects segregated across each disease] ........................................................................................................................................................................................................37
Figure 4.18. Execution output2 for use case [Patients who had moderate side effects segregated across each disease] ........................................................................................................................................................................................................38
Figure 4.19. Output result for use case [Patients having moderate side effects segregated across each disease] ........................................................................................................................................................................................................39
Figure 4.20. Execution output1 for use case [Partitioning whole data set based on the states] ........................................................................................................................................................................................................40
Figure 4.21. Execution output2 for use case [Partitioning whole data set based on the states] ........................................................................................................................................................................................................40
Figure 4.22. Output result1 for use case [Partitioning whole data based on the states] ........................................41
Figure 4.23. Output result2 for use case [Partitioning whole data based on the states] ........................................41
Figure 4.24. Output result3 for use case [Partitioning whole data set based on the states] ........................................................................................................................................................................................................42
Figure 4.25. Execution output1 for use case [Percentages of insurance company usage per year for a given state] ........................................................................................................................................................................................................43
Figure 4.26. Execution output2 for use case [Percentages of insurance company usage per year for a given state] ........................................................................................................................................................................................................43
Figure 4.27. Output result1 for use case [Percentages of insurance company usage per year for a given state] ........................................................................................................................................................................................................44
Figure 4.28. Output result2 for use case [Percentages of insurance company usage per year for a given state] ........................................................................................................................................................................................................45
Figure 4.29. Execution output1 for use case [Company that supplied most medications in each year] ........................................................................................................................................................................................................46
Figure 4.30. Execution output2 for use case [Company that supplied most medications in each year] ........................................................................................................................................................................................................47
Figure 4.31. Output result for use case [Company that supplied most medications in each year] ........................................................................................................................................................................................................47
Figure 4.32. Table creation in Hive. .................................................................48
Figure 4.33. loading input file to Table. ..........................................................48
Figure 4.34. Execution output for use case using Hive [Patients who had moderate side effects segregated across each state]. ........................................................................49
Figure 4.35. Output result for use case using Hive [Patients who had moderate side effects segregated across each state]. ........................................................................50
Figure 4.36. Execution output for use case using Hive [Patients in each blood group across each state] ...........................................................................51
Figure 4.37. Output result for use case using Hive [Patients in each blood group across each state]. ...........................................................................52
Figure 4.38. Pig execution in HDFS mode. .......................................................53
Figure 4.39. Load data using Pig. .................................................................53
Figure 4.40. Execution output1 for use case using Pig [Patients in each blood group]........54
Figure 4.41. Execution output2 for use case using Pig [Patients in each blood group]........54
Figure 4.42. Output result for use case using Pig [Patients in each blood group]...........55
Figure 4.43. Execution output1 for use case using Pig [Patients needing blood in each state]. ...........................................................................55
Figure 4.44. Execution output2 for use case using Pig [Patients needing blood in each state]. ...........................................................................56
Figure 4.45. Output result for use case using Pig [Patients needing blood in each state]. ...........................................................................56
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract, Transform and Load</td>
</tr>
<tr>
<td>BI</td>
<td>Business Intelligence</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>WinSCP</td>
<td>Windows Secure Copy</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to my advisor Prof. Carl Eckberg for all his help and support in completing my thesis.

I would also like to thank Prof. Alan Riggins and Prof. Carmelo Interlando for serving as committee members of the thesis.

Lastly, I would like to thank my husband and parents for their constant encouragement and support in finishing my thesis.
CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

RDBMS (Relational Data Base Management System) became dominant in the year 1980 and earlier models disappeared. They use a relational model. The RDBMS model maintains relationships between tables using primary keys, foreign keys and indexes. RDBMS also allows us to keep data unique by avoiding redundancy and is much more secure [1]. But RDBMS’s still have inefficiency problems with vast amounts of data.

Day to day the amount of data that’s being stored is exponentially increasing. Due to the internet and mobile revolution, every day a huge amount of data is getting generated. The new raw data that’s getting generated can be wireless network sensor data / web server data / RF ID reader data/ cache register data / network equipment data / system logs, etc.

ETL (Extract, Transform, and Load) jobs can be run on the raw data in order to extract the raw data, transform the data into a proper format and store the data in an RDBMS. In some cases, BI (Business Intelligence) tools can use the data from RDBMS for performing data analytics and generating reports.

The block diagram illustrating the Data collection / Storage to RDBMS / Data analytics using BI tools is provided in Figure 1.1.

RDBMS requires pre-processing of data using separate ETL process before one can use it. But this processing approach is limited in its success, especially with very large amounts of unstructured data.

But today many organizations are using Hadoop when operating with large data sets. Hadoop is much more flexible and it doesn’t require pre-processing of data using separate ETL processes as in parallel RDBMS. Hadoop is also more economical and scalable compared with RDBMS [2].
Figure 1.1. Data Analytics using Traditional Systems. Source: [3]
1.2 Thesis Contribution

A 1.5 Gb size health care data set is considered for data analytics. The data set comprises of 1.5 million patients health records. Different use cases have been considered for data analytics which are accomplished using MapReduce, Hive and Pig functionalities of Hadoop. For data analytics use cases, the data has been partitioned state wise as an intermediate step to help reduce the data processing time. The data was generated so as to be realistic, since actual data is not public as detailed in section 4.1.1.

1.3 Thesis Organization

Chapter 2 provides an overview on Big Data. The Hadoop architecture is described in detail. The Hadoop extensions - Hive and Pig are explained in detail.

Chapter 3 provides the step by step procedure for Hadoop installation. A Virtual Machine (VM) setup procedure has been described. The procedure to move data from VM to HDFS (Hadoop Distributed File System) has been detailed.

Chapter 4 provides information on the fields of the health care data set used for data analytics. It provides details on the tool that is used for realistic mock data set generation. Different data analytics use cases are discussed, the implementation is explained and the use case outputs have been listed in this chapter.
CHAPTER 2

BIG DATA AND HADOOP

2.1 BIG DATA

Big Data is a term used to suggest huge data sets (several gigabytes / terabytes / petabytes) of data. The data is so large and complex that it would become difficult to process using traditional data processing applications. Big data requires a new set of tools, applications and frameworks to process and manage data.

2.1.1 Evolution of Data/Big Data

Data has always been around and there has been a need to store, process and manage data since the beginning of modern human civilization. However, the amount of data captured, stored, processed and managed depends on various factors including the necessity felt by humans for certain information, available tools and technologies needed for making decisions based on the data analysis and so on.

In today’s world, due to advancements of technology there is a huge (several terabytes /petabytes) amount of data that’s being constantly captured [4]. Natural curiosity about truly important things, like whether more teenagers like Justin Bieber than millennials, demand processing Twitter data, which is huge.

2.1.2 Characteristics of Big Data

The characteristics of Big Data are popularly known as the three V’s of Big Data [4] (see Figure 2.1).
2.1.2.1 VOLUME

Volume refers to the size of the data that the user is working with. Due to advancements of technology the amount of data that is being generated is growing rapidly. Data is spread across different places, different formats, in large volumes ranging from gigabytes to terabytes to petabytes. Data is not only being generated from humans but by machines too. Now-a-days the data that’s getting generated from machines is surpassing that data that’s being generated by Humans [4]. Weather data is a good example.

2.1.2.2 VARIETY

Variety refers to different formats in which data is getting generated. Apart from structured data like spreadsheets and traditional flat files, there is a large amount of unstructured data that’s being generated in the form of weblogs, sensor data, social media,
etc. Enterprises are making use of both structured and unstructured data for data analysis, and thereby making better business decisions to stay competitive [4].

2.1.2.3 Velocity

Velocity refers to the speed at which data is getting generated. Different applications in different fields have different requirements. So we see data getting generated at different speeds based on the application requirements [4].

2.2 Hadoop

Hadoop is an open source framework. It is capable of processing large amounts of data sets in a distributed fashion across clusters using a simplified programming model. Hadoop provides a reliable way to store, process and analyze the data [4].

2.2.1 Hadoop Architecture

The Hadoop architecture is illustrated in Figure 2.2.
Hadoop works in a master-slave fashion. Hadoop has two core components-
HDFS (Hadoop Distributed File System) and MapReduce.

2.2.2 Hadoop Components

2.2.2.1 HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

HDFS offers a reliable and distributed storage. It replicates the data across multiple
nodes on clouds or commodity computers. Unlike a regular file system, when data is pushed
into HDFS, it internally splits into multiple data blocks (configurable parameter with default
size of 64Mb). Each incoming file is broken into 64Mb data block by default and all blocks
which make up the file are of the same size 64Mb except the last block which could be less
than 64Mb depending upon the size of the incoming file.

HDFS also replicates (replication rate is configurable) the data across various data
nodes thus ensuring fault tolerance and reliability. It also ensures that a replication factor is
maintained, so that if a node goes down, one can recover by using replicated data on other
nodes.

HDFS is capable of storing large amounts of data which can be structured or un-
structured. The computers present in the cluster can be present in any location and there is no
physical location dependency [4].

HDFS works in a master/slave fashion.

NameNode: NameNode is a master component and holds the information about all
other nodes in the Hadoop cluster, files present and their locations in the cluster.
There is only one NameNode per cluster.

DataNode: DataNode is a slave node and holds the user data in the form of data
blocks. There can be a lot of DataNodes in a Hadoop cluster.

2.2.2.2 MAPREDUCE

MapReduce offers a framework/analysis system which performs complex
computations on large datasets in a parallelized fashion. This system breaks down the
complex computations into multiple smaller tasks and assigns those to individual slave nodes
and takes care of the co-ordination and consolidation of the results [4]. These tasks run
independently on various nodes across the cluster. There are primarily two types of tasks:
Map tasks and Reduce tasks.
As in HDFS, MapReduce (computation part) also works in master/slave fashion.

**JobTracker:** Keeps track of the tasks assigned and co-ordinates the exchange of information and the results with the slave nodes. Its responsibility also includes rescheduling of failed tasks and monitoring the overall progress of the job. There is only one JobTracker per cluster.

**TaskTracker:** Acts as a slave and is responsible for running the tasks assigned by the JobTracker and providing the results back to the JobTracker. There can be multiple TaskTracker nodes that can exist in a cluster.

Data processing using the MapReduce framework is highlighted in Figure 2.3.

**FileInputFormat:** This is the input file/data which need to be processed.

**Split:** Hadoop splits the incoming data into several blocks.
**RecordReader:** RecordReader helps to read the data line by line and converts into key/value pairs to be passed as the input to the Mapper.

**Mapper:** Mapper contains the logic to process input data. The Map function transforms the input records to intermediate records.

**Combiner:** This is an optional step often used to improve the performance by reducing the data to be transferred across the network [4].

**Shuffle:** Output of all the mappers is collected, shuffled and sorted, to be sent to the Reducer.

**Reducer:** Reducer applies logic to aggregate the data and provide it to an FileOutputFormat class.

**FileOutputFormat:** It is a pre-defined class provided by the MapReduce framework through which final output can be written to HDFS.

### 2.2.3 Hadoop Characteristics

- Hadoop provides a reliable shared storage system (HDFS) and data analysis system (MapReduce) [4].
- Cost effective, as it can work with commodity hardware and doesn’t need expensive hardware.
- Flexible and can process both structured as well as un-structured data sets.
- Is optimized for large and very large data sets. It takes a lot less data processing time due to parallel processing, when compared with traditional data base management systems.
- Very scalable. As a result the Hadoop cluster can contain hundreds or thousands of servers.
- Provides a very reliable system as data is replicated across multiple nodes (replication factor is configurable).

### 2.2.4 Hive

Hive is a data warehouse infrastructure which is built on top of the Hadoop distributed system and it provides tools to enable easy ETL to join, aggregate and filter different data sets. It also allows programmers to build custom MapReduce functionalities. Hive provides an SQL like query interface called HiveQL which internally does MapReduce operations. Hive is extremely useful when processing large amounts of data (terabytes). Hive is easier to use as it abstracts the complexity of Hadoop. Lots of companies support Hive, a simple reason being to encourage SQL based queries on top of Hadoop.
2.2.4.1 HIVE ARCHITECTURE

The Hadoop Hive Architecture diagram is presented in Figure 2.4.

When a user logs in to Hive terminal through a CLI (command line interface) or a Web graphical user interface, it directly connects to Hive drivers through the Thrift server. The queries which are written by users are received by drivers and sent to Hadoop, where Hadoop gets the data and divides the work using NameNode, DataNode, JobTracker and TaskTracker.
2.2.4.2 HIVE COMPONENTS

2.2.4.2.1 Thrift server
This component is optional. This allows a remote client to submit requests to Hive to retrieve results. A variety of programming languages can be used to accomplish this [5].

2.2.4.2.2 Driver
Driver is a very important component that takes all the requests from CLI (command line interface) or a web interface, or the Thrift server, and does the compilation, optimization and execution of the data.

2.2.4.2.3 Meta Store
This component stores all the structure information of various tables and partitions in the warehouse including column and column type information, serializers and de-serializers necessary to read and write data and corresponding HDFS files where the data is stored [6].

2.2.4.2.4 HDFS
All data is stored in HDFS. A detailed explanation of HDFS has been provided in section 2.2.2.1. Hive currently uses HDFS as its execution engine.

2.2.4.3 Disadvantages of Hive
- It’s not designed for online transaction processing.
- There is a built-in latency for every job.
- When Hive compiles a query into a set of MapReduce jobs it has to co-ordinate and launch the jobs on the cluster.

2.2.5 Pig
Pig is a high level data flow system that provides a simple language popularly known as Pig Latin that can be used for manipulating data and queries. Pig Hadoop was developed by Yahoo in the year 2006 such that they can have an ad-hoc method for creating and executing Map Reduce jobs on huge data sets [7]. Pig has relational database features and is built on top of Hadoop and makes it easier to clean and analyze big data sets without having to write vanilla MapReduce jobs in Hadoop [8].
The Pig tool itself converts all high level operations into MapReduce jobs. It follows a multi query approach and helps cut down the number of times the data is scanned. Performance of Pig is on par with the performance of raw MapReduce. The Pig programs structure is amenable to substantial parallelization which enables them to handle very large data sets [9]. Pig could be used for ETL tasks naturally as it can handle unstructured data.

2.2.5.1 PIG ARCHITECTURE

The Pig Hadoop architecture diagram is presented in Figure 2.5.

![Figure 2.5. Pig Architecture block diagram.](image)

2.2.5.2 PIG LATIN COMPILER

The Pig Latin compiler converts the Pig Latin code into executable code. The executable code is in the form of MapReduce jobs [10].

The sequence of MapReduce programs enables Pig programs to do data processing and analysis in parallel.

2.2.5.3 BENEFITS OF PIG

- Learning curve is not steep.
- Decrease in development time when compared with the vanilla MapReduce jobs due to reduced complexity and maintenance needs [8].
• Helps with faster prototyping of algorithms due to the ease of using the Pig Latin language.
• Effective for unstructured data.
• It’s procedural. Provides better expressiveness in the transformation of data at every step [8].

2.2.5.4 DISADVANTAGES OF PIG

• It is not very mature. Even though it has been around for quite some time, it’s still in development.
• Doesn’t clearly distinguish the type of error. It just gives an execution error when something goes wrong. Doesn’t specify if it’s a syntax error or run time error or type error.
• Support: Google and Stackoverflow doesn’t generally lead to good solutions for problems [8].
• Typically for complex business logic involving encryption of data, Pig is not used. Java API for cryptography is picked over Pig in such cases.
CHAPTER 3

HADOOP INSTALLATION

Hadoop Installation on Windows PC has been done for the data analytics implementation. The sections below provide the details on installation and storage of data in HDFS.

3.1 VIRTUAL MACHINE INSTALLATION

Installation of Cloudera VMware virtual machine is very useful to support the Hadoop framework.

Minimum requirements for virtual machine installation on windows are

- 4GB RAM
- Dual Core processor
- 20 GB free space in hard disk.

The step by step procedure that needs to be followed is

**Step 1**: Download Open source VMWare player from the link below
https://download3.vmware.com/software/player/file/VMware-player-7.1.2-2780323.exe

**Step 2**: Download Cloudera VMware virtual machine from the link below
http://www.cloudera.com/downloads/quickstart_vms/5-5.html

After downloading and installing both the VMware player and the Cloudera virtual machine, the following welcome screen would be seen (see Figure 3.1).

These steps would need to be followed to work with the virtual machine.

**Step 1**: Click on the “Open a Virtual Machine” option as shown in Figure 3.1.

**Step 2**: Select cloudera-quickstart-vm-5.5.0-0-vmware.vmx from downloads and click open.

**Step 3**: After selecting a virtual machine, the hardware settings of the virtual machine can be changed if needed. For example- in the settings shown in Figure 3.2, the allocated RAM for VM has been changed to 8GB so that the VM can run faster.
Figure 3.1. Virtual Machine installation.
Figure 3.2. Virtual Machine settings.

**Step 4:** After clicking OK in Figure 3.2, the user will be able to click “Play Virtual Machine”
3.2 Loading Data into HDFS

To load data from a virtual machine to HDFS, first the data set needs to be copied from the local machine to the virtual machine using WinSCP (Windows secure copy) software.

Step 1: Loading data from local machine to virtual machine by using WinSCP.

WinSCP is open source software which can be downloaded from the link below http://winscp.en.softonic.com/

The result of executing this download is Figure 3.5.

To login into the Cloudera virtual machine through WinSCP, we need the username and password of the virtual machine to transfer data. We can find the IP address of the virtual
machine as shown below, using a windows command window, which is obtained by clicking on the command icon on the toolbar at the top of Figure 3.3.

![Figure 3.4. Finding IP address of VM.](image)

WinsSCP need the following credentials to connect to VM - Username -cloudera /Password-cloudera and IP address-192.168.175.128.
Step 2: Loading data from Cloudera virtual machine to HDFS

In the virtual machine we would need to check if the data is present or not before starting the process for loading data to HDFS as shown in Figure 3.6.
Figure 3.6. Data sanity check in VM before moving to HDFS.

Following are simple Hadoop Commands that are used to perform different operations on HDFS

$ hadoop version: To find the version of Hadoop that’s in use.

$hdfs dfs -mkdir /HealthCare: Create new directory with the name HealthCare in HDFS.

$hdfs dfs -rmr /directory_path: Delete the directory which is located in the specified path from HDFS.

$hdfs dfs -rm /file_path: Delete the file in HDFS located in the specified path.

$hdfs dfs -ls /: List all the files and directories in HDFS.

$hdfs dfs put /src_path /dest_path: Copies source files from local file system to destination file system.

Following are the steps to move the data to HDFS

**Step1:** create a directory in HDFS

$ hadoop dfs -mkdir /user/cloudera/HealthCare/Input

**Step2:** Copy data set from local machine to HDFS by

$ hadoop fs -put Healthcare15.json /user/cloudera/HealthCare/Input as in Figure 3.7.
After copying dataset into HDFS, a sanity check can be performed to check for the data presence in the HDFS file browser as shown below. The steps are

**Step1:** In Figure 3.3, click the browser icon in the top toolbar.

**Step2:** Click on Hue to open a Hue toolbar.

**Step3:** Click on “File Browser” in the Hue toolbar. This will display as seen in Figure 3.8.
Figure 3.8. Data sanity check in HDFS.
CHAPTER 4

HEALTH CARE DATA ANALYTICS

In this thesis, a health care data set is considered for data analytics using Hadoop technologies.

4.1 HEALTH CARE DATA SET

The 1.5 GB health care data set comprises of 1.5 million patient records collected over the period of 6 years from 2010 to 2015. This is a realistic mock data set generated by using an online tool at www.mockaroo.com [11].

4.1.1 Data Set Generation

Healthcare data set is generated through the online tool at www.mockaroo.com. This is a mock data set and using the custom features provided in the tool, the data is made as realistic as possible. The data set is generated in JSON format and contains the following fields.

A detailed description of each field in the data set is provided in Table 4.1.
Table 4.1. Health Care Data Set Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description of the field and its scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>First name of the patient</td>
</tr>
<tr>
<td>Last Name</td>
<td>Last name of the patient</td>
</tr>
<tr>
<td>Age</td>
<td>Age of the patient. Scope: 20 - 64</td>
</tr>
<tr>
<td>Gender</td>
<td>Gender of the patient</td>
</tr>
<tr>
<td>Race</td>
<td>Race of the patient</td>
</tr>
<tr>
<td>Email_id</td>
<td>Email id of the patient</td>
</tr>
<tr>
<td>Job_Title</td>
<td>Job title of the patient</td>
</tr>
<tr>
<td>Company_Name</td>
<td>Patient's employer</td>
</tr>
<tr>
<td>Street_Address</td>
<td>Patient's address(Street)</td>
</tr>
<tr>
<td>City</td>
<td>Patient's address (City)</td>
</tr>
<tr>
<td>State</td>
<td>Patient's address (State)</td>
</tr>
<tr>
<td>Country</td>
<td>United States</td>
</tr>
<tr>
<td>Date_of_Hospital_visit</td>
<td>Date on which patient visited the hospital</td>
</tr>
<tr>
<td>Temperature</td>
<td>Temperature of the patient during the hospital visit</td>
</tr>
<tr>
<td>Blood_group</td>
<td>Blood group of the patient. Scope :O+, O-, A+, A-, B+, B-, AB+, AB -</td>
</tr>
<tr>
<td>Blood_Transfusion Needed</td>
<td>Whether person needed blood transfusion</td>
</tr>
<tr>
<td>Diagnosis_description</td>
<td>Description of the disease patient is diagnosed with</td>
</tr>
<tr>
<td>Procedure_description</td>
<td>Medical procedure performed on the patient</td>
</tr>
<tr>
<td>Drug_Name</td>
<td>Medicine used by the patient</td>
</tr>
<tr>
<td>Drug_Company</td>
<td>Pharmaceutical company that supplied the medication</td>
</tr>
<tr>
<td>Side_Effects</td>
<td>Side effects due to medicine intake</td>
</tr>
<tr>
<td>Insurance_Provider</td>
<td>Patient's insurance provider</td>
</tr>
<tr>
<td>Insurance_Claimed</td>
<td>Amount paid by the insurance provider. Scope $800 - $6000</td>
</tr>
</tbody>
</table>

www.mockaroo.com provides the flexibility to customize and define the scope of each of the fields to help make the dataset it generates as realistic as possible. This tool provides a handle to the user to specify formulas, regular expressions and also define the scope of a field to a custom list during data generation. A few examples have been listed below to illustrate how the customization was done during the data generation. These are complex parameters used by mockaroo when it generates the very large dataset.
- **Blood group**: Blood group can take one of the following: O+, O-, A+, A-, B+, B-, AB+, AB-. The percentages have been specified in the custom distribution to ensure the data is realistic i.e., 35% of patients would be with O+ blood group followed by 25% of patients with A+ blood group, etc.

  ![Figure 4.1. Custom list creation for blood group data. Source: [11]](image)

  The parameter above will cover 35% of the 1.5 million mock patient records to have the blood type O+, etc.

- **Insurance_Claimed**: Amount paid by the insurance company is taken as a random value from $800 to $6000. Its generated using mockaroo’s in built “random” API.

  ![Figure 4.3. In built formula for “Insurance_Claimed” field generation. Source: [11]](image)

- **Side_Effects**: A regular expression is used for generating the data associated with the field Side_Effects.

  ![Figure 4.4. Regular Expression used for “Side_Effects” field generation. Source: [11]](image)

A snippet of the final data set in JSON (JavaScript Object Notation) format is as shown in Figure 4.5.
4.2 HEALTH CARE DATA ANALYTICS

Different use cases have been considered, to perform data analytics on the health care data set. Details on different use cases and the implementation details, have been provided below. Data analytics outputs have also been provided below.

4.2.1 Patients in Each Blood Group Across Each State

MapReduce framework has been used and the implementation is done in Java. Hadoop pre-defined Mapper and Reducer packages have been imported to accomplish this.

A configuration object has been created to get the details of block size, replication factor, NameNode, JobTracker and TaskTracker information which are predefined in the configuration class. This information is used in the future to split and process the data. The input file path has been passed as an argument. In the MapReduce framework, the FileInputFormat class is a predefined class which needs information on the data format of the input file. In this case, the input file format is in text. Internally the FileInputFormat class
takes the input file from HDFS and splits the input file into blocks known as “splits” based on the default block size. RecordReader reads the input file line by line and converts it into key/value format and provides it as an input to Mapper. Three arguments: key, value and context would be passed to the Mapper. The Key is in long writable format which is an offset of each row/line from the start of the file, and the value is in text data type. The whole JSON row/line is considered as the value. The third argument, context, is an object of the Context class. Mapper uses a pre-defined Context class to write the mapper output.

In the mapper class, for each row, the “blood_group” and “State” are fetched and written as the key using context write. The value for this key entry is given as 1 which reflects the count.

The Reducer class also takes three arguments: key, value and the context. Reducer performs shuffle and sort operations internally and aggregates all the values corresponding to each key. The aggregated value along with key gets written using a pre-defined Context object. The FileOutputFormat class helps in writing the final output to HDFS.

To run and retrieve the results, the Java file is converted into a jar file and will run on cluster to get the results.

The steps used to create the jar file in Eclipse are as shown below:

Go to File → Export → Java → JAR File → select HealthCareDataAnalytics→Give Jar file name→Click next→Finish

Figure 4.6 shows a screenshot of steps in the Eclipse creation of the jar file.
The command in the VM to execute the jar file whose purpose is discussed in section 4.2.1 is as follows:

```bash
$ hadoop jar HealthCareDataAnalysis.jar com.sdsu.main.PatientsBloodGroupEachState 
/user/cloudera/Healthcare/Input 
/user/cloudera/Healthcare/Output/new_dataset_number_of_patients_in_each_blood_group_across_each_state
```

This can be seen in Figure 4.7.
Figure 4.7. Execution output for use case [Patients in each blood group across each state].

The two screenshots in Figures 4.8 and 4.9 show the output.
Figure 4.8. Output1 for use case [Patients in each blood group across each state].

<table>
<thead>
<tr>
<th>State</th>
<th>Number of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia</td>
<td>5668</td>
</tr>
<tr>
<td>Washington</td>
<td>3677</td>
</tr>
<tr>
<td>West Virginia</td>
<td>3529</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>1744</td>
</tr>
<tr>
<td>Wyoming</td>
<td>83</td>
</tr>
<tr>
<td>Alabama</td>
<td>1748</td>
</tr>
<tr>
<td>Alaska</td>
<td>344</td>
</tr>
<tr>
<td>Arizona</td>
<td>1979</td>
</tr>
<tr>
<td>Arkansas</td>
<td>421</td>
</tr>
<tr>
<td>California</td>
<td>8772</td>
</tr>
<tr>
<td>Colorado</td>
<td>1835</td>
</tr>
<tr>
<td>Connecticut</td>
<td>1006</td>
</tr>
<tr>
<td>Delaware</td>
<td>319</td>
</tr>
<tr>
<td>District of Columbia</td>
<td>2598</td>
</tr>
<tr>
<td>Florida</td>
<td>6998</td>
</tr>
<tr>
<td>Georgia</td>
<td>2297</td>
</tr>
<tr>
<td>Hawaii</td>
<td>302</td>
</tr>
<tr>
<td>Idaho</td>
<td>392</td>
</tr>
<tr>
<td>Illinois</td>
<td>965</td>
</tr>
</tbody>
</table>

Figure 4.9. Output2 for use case [Patients in each blood group across each state].

<table>
<thead>
<tr>
<th>State</th>
<th>Number of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>8952</td>
</tr>
<tr>
<td>Alaska</td>
<td>1748</td>
</tr>
<tr>
<td>Arizona</td>
<td>8495</td>
</tr>
<tr>
<td>Arkansas</td>
<td>2881</td>
</tr>
<tr>
<td>California</td>
<td>43781</td>
</tr>
<tr>
<td>Colorado</td>
<td>9289</td>
</tr>
<tr>
<td>Connecticut</td>
<td>5400</td>
</tr>
<tr>
<td>Delaware</td>
<td>1469</td>
</tr>
<tr>
<td>District of Columbia</td>
<td>12066</td>
</tr>
<tr>
<td>Florida</td>
<td>33228</td>
</tr>
<tr>
<td>Georgia</td>
<td>11668</td>
</tr>
<tr>
<td>Hawaii</td>
<td>1823</td>
</tr>
<tr>
<td>Idaho</td>
<td>1830</td>
</tr>
<tr>
<td>Illinois</td>
<td>957</td>
</tr>
</tbody>
</table>
4.2.2 Patients Needing Blood in Each State for Each Year

Patient’s “Date_of_Hospitalization” field is used to extract the “Year”. The Mapper has the implementation to take the passed input value (JSON row) and based on the “Blood_Transfusion_Needed” input, the “blood_group”, “State” and “Year” are written as the key using a Context object. The Reducer would later aggregate the values and write the aggregated value using a Context object. The FileOutputFormat class later writes the final output to HDFS.

The commands needed to acquire the outputs are:

hadoop jar HealthCareDataAnalysis.jar com.sdsu.main.PatientsNeedBloodInEachState /user/cloudera/HealthCare/Input
/user/cloudera/HealthCare/Output/new_dataset_number_of_patients_need_blood_in_each_state

The commands are visible in Figure 4.10.

Figure 4.10. Execution output1 for use case [Patients needing blood in each state for each year].

The resulting feedback from these commands is shown in Figure 4.11.
Figure 4.11. Execution output2 for use case [Patients needing blood in each state for each year].

The two screenshots in Figures 4.12 and 4.13 show the output.
Figure 4.12. Output1 for use case [Patients needing blood in each state for each year].

Figure 4.13. Output2 for use case [Patients needing blood in each state for each year].
4.2.3 Disease Affecting the Most for Each Race

The Mapper has been implemented to take a JSON row as the input and write the “Race” as the key using a Context object. The “Disease_diagnosed” and count, which is 1, are written as the value. Note that here “Race” is more usually described as “nationality” or “place of origin”.

In the Reducer, another hash table is initialized and the values passed to the reducer is converted and written into this new hash table. The new hash table would contain the “Disease_diagnosed” as the key, and the sum of all the counts would become the value corresponding to that key. Later a logic is written to find the entry in the new hash table which has the maximum value, which effectively provides the disease that affects the most people for each race. Later the output is written to HDFS using FileOutputFormat class.

The commands needed to acquire the outputs are:

$ hadoop jar HealthCareDataAnalysis.jar com.sdsu.main.RaceEffectedDisease /user/cloudera/HealthCare/Input /user/cloudera/HealthCare/Output/new_dataset_disease_effected_for_each_race_people

The commands are visible in Figure 4.14.

The resulting feedback from these commands is shown in the Figure 4.15.
Figure 4.15. Execution output2 for use case [Disease affecting the most for each race].

The screenshot in Figure 4.16 shows the output.
4.2.4 Patients Who Had Moderate Side Effects Segregated Across Each Disease

The Mapper has the implementation to take the passed input value (JSON row) and based on the “Side_Effects” input, the disease is written as the key using a Context object. The Reducer would later shuffle, sort and aggregate the values and write the aggregated value using a Context object. The FileOutputFormat class later writes the final output to HDFS.

The commands needed to acquire the output are:

```
$ hadoop jar HealthCareDataAnalysis.jar
com.sdsu.main.PatientsEachCategoryWithModerateSideEffects
/user/cloudera/HealthCare/Input
/user/cloudera/HealthCare/Output/new_dataset_patients_had_moderate_side_effects_for_each_disease
```
The commands are visible in Figure 4.17 below.

Figure 4.17. Execution output1 for use case [Patients who had moderate side effects segregated across each disease].

The resulting feedback from these commands is shown in Figure 4.18.
Figure 4.18. Execution output2 for use case [Patients who had moderate side effects segregated across each disease].

The screenshot in Figure 4.19 shows the output.
4.2.5 Partitioning Whole Data Set Based on States

To partition the whole healthcare dataset based on states, three classes have been created under the partition package, which are Driver, Mapper and Reducer classes. The Mapper has the implementation to take the passed input value (JSON row) and the “State” is written as the key and the input JSON row is written as the value using a Context object.

The Reducer has the implementation to use the MultipleOutputs class, create file names based on state names and write the data based on the states.

The commands needed to acquire the outputs are:

$ hadoop jar HealthCareDataAnalysis.jar com.sdsu.partition/PartitionerDriver
/user/cloudera/HealthCare/Input
/user/cloudera/HealthCare/Output/new_dataset_Input_statewise_partition

The commands are visible in the Figure 4.20.
Figure 4.20. Execution output1 for use case [Partitioning whole data set based on the states].

The resulting feedback from these commands is shown in Figure 4.21.

Figure 4.21. Execution output2 for use case [Partitioning whole data set based on the states].

The three screenshots in Figures 4.22, 4.23 and 4.24 show the output.
Figure 4.22. Output result1 for use case [Partitioning whole data based on the states].

Partitioned data of Alaska state is shown in Figure 4.23.

Figure 4.23. Output result2 for use case [Partitioning whole data based on the states].

Partitioned data of California state is shown in Figure 4.24.
4.2.6 Percentages of Insurance Company Usage Per Year for a Given State

A Mapper has been implemented to take a JSON row of the partitioned data for a state as the input and write the “year” as the key using a Context object. This is found using “Date_of_Hospitalization” field. The value is more complicated. For example, if Aetna appears in five rows in 2014, then one of the values for 2014 will be (Aetna, (1, 1, 1, 1, 1)). The number of Aetna insurers will be totaled later. In the Reducer, another hash table is initialized and the values passed to the reducer are converted and written into this new hash table. The new hash table would contain the “Insurance_Provider” as the key and the sum of all the counts would become the value corresponding to that key. Later a logic is written to find the percentage based on the value for a given key and the value in the hash table is updated with the percentage. Later the output is written to HDFS using the FileOutputFormat class.

The commands needed to acquire the output are:

```shell
$ hadoop jar HealthCareDataAnalysis.jar com.sdsu.main.InsurancePercentage /user/cloudera/HealthCare/Output/new_dataset_Input_statewise_partition /user/cloudera/HealthCare/Output/new_dataset_percentage_of_insurance_claimed_by_each_insurance_company Florida
```

The commands are visible in Figure 4.25.
Figure 4.25. Execution output1 for use case [Percentages of insurance company usage per year for a given state].

The resulting feedback from these commands is shown in the Figure 4.26.

Figure 4.26. Execution output2 for use case [Percentages of insurance company usage per year for a given state].

The two screenshots in Figures 4.27 and 4.28 show the output of this use case.
Figure 4.27. Output result1 for use case [Percentages of insurance company usage per year for a given state].
4.2.7 Company That Supplied Most Medications in Each Year

The Mapper has been implemented to take a JSON row as the input and write the “Year” as the key using a Context object. This is found using “Date_of_Hospitalization” field. The value is more complicated. For example, if Walmart appears in five rows in 2014, then one of the values for 2014 will be (Walmart, (1, 1, 1, 1, 1)). The number of medications supplied by Walmart would be totaled later.

In the Reducer, another hash table is initialized and the values passed to the reducer are converted and written into this new hash table. The new hash table would contain the “Drug_Company” as the key and the sum of all the counts would become the value corresponding to that key. Later a logic is written to find the entry in the new hash table which has the maximum value which effectively provides the drug company that supplied the most medications for a given year. Later the output is written to HDFS using the FileOutputStream class.

The commands needed to acquire the outputs are

Figure 4.28. Output result2 for use case [Percentages of insurance company usage per year for a given state].
$ hadoop jar HealthCareDataAnalysis.jar com.sdsu.main.TopDrugCompaniesInYear
/user/cloudera/HealthCare/Input
/user/cloudera/HealthCare/Output/new_dataset_top_drug_company_in_each_year

The commands are visible in Figure 4.29.

![Hadoop Job Output](image)

**Figure 4.29. Execution output1 for use case [Company that supplied most medications in each year].**

The resulting feedback from these commands is shown in Figure 4.30.
Figure 4.30. Execution output2 for use case [Company that supplied most medications in each year].

The screenshot in Figures 4.31 shows the output.

Figure 4.31. Output result for use case [Company that supplied most medications in each year].
4.2.8 Patients Who Had Moderate Side Effects Segregated Across Each State [Implemented Using Hive]

Hive abstracts the complexity of Hadoop. Hive provides an SQL like query interface called HiveQL which internally does MapReduce operations.

A Hive SQL query was used to accomplish this use case. First a table is created in Hive using “CREATE TABLE” statement and input data file is loaded into the table using “LOAD DATA” statement.

Creation of table in Hive is shown in Figure 4.32.

![Figure 4.32. Table creation in Hive.](image1)

Loading the input file to table is shown in Figure 4.33.

![Figure 4.33. loading input file to Table.](image2)

The following Hive query was used and can be seen in Figure 4.34.
select state, Side_Effects, count(Side_Effects) from HealthCareData where Side_Effects like "%Moderate%" group by state, Side_Effects

The resulting feedback from this Hive query is shown in Figure 4.34.

Figure 4.34. Execution output for use case using Hive [Patients who had moderate side effects segregated across each state].

The screenshot in Figures 4.35 shows the output.
Figure 4.35. Output result for use case using Hive [Patients who had moderate side effects segregated across each state].

4.2.9 Patients in Each Blood Group Across Each State
[Implemented Using Hive]

The following Hive query was used

```sql
select state, blood_group, count(blood_group) from HealthCareData
group by blood_group, state
```

The resulting feedback from this Hive query is shown in Figure 4.36.
Figure 4.36. Execution output for use case using Hive [Patients in each blood group across each state].

The screenshot in Figures 4.37 shows the output.
4.2.10 Patients in Each Blood Group [Implemented Using Pig]

Pig provides a simple language popularly known as Pig Latin that can be used that for manipulating data and queries. The Pig tool itself converts all high level operations into MapReduce jobs.

Pig Scripts can be executed either on Hadoop cluster or in local mode (running on VM).

To execute Pig scripts on MapReduce or HDFS mode the following command can be used

```
pig -x mapreduce
```

To execute Pig scripts on local mode following command can be used.
pig -x local

Pig execution is shown in Figure 4.38.

![Figure 4.38. Pig execution in HDFS mode.](image)

Loading the data using Pig is shown in Figure 4.39.

![Figure 4.39. Load data using Pig.](image)

A Pig script was used to accomplish this use case.

The Pig script is visible in Figure 4.40.
Figure 4.40. Execution output1 for use case using Pig [Patients in each blood group].

The resulting feedback from this Pig script is shown in Figure 4.41.

Figure 4.41. Execution output2 for use case using Pig [Patients in each blood group].

The screenshot in Figures 4.42 shows the output.
4.2.11 Patients Needing Blood in Each State
[Implemented Using Pig]

A Pig script has been implemented to accomplish this use case. Data from HDFS is loaded and the Pig script is executed to list the patients needing blood in each state.

The Pig script is visible in Figure 4.43.
The resulting feedback from this Pig script is shown in Figure 4.44.

Figure 4.44. Execution output for use case using Pig [Patients needing blood in each state].

The screenshot in Figures 4.45 shows the output.

Figure 4.45. Output result for use case using Pig [Patients needing blood in each state].
BIBLIOGRAPHY

WORKS CITED


WORKS CONSULTED

HOUSTON CHRONICLE, *Small business*. Hearst Newspapers,  

M. KERZNER AND S. MANIYAM, *Hadoop illuminated*. Hadoop Illuminated,  