5-2015

Using Hadoop to Support Big Data Analysis: Design and Performance Characteristics

Afreen Sultana
St Cloud State University, asultana@stcloudstate.edu

Using Hadoop to Support Big Data Analysis: Design and Performance Characteristics

by

Afreen Sultana

A Starred Paper

Submitted to the Graduate Faculty

of

St. Cloud State University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Information Assurance

May, 2017

Starred Paper Committee:
Dr. Dennis Guster, Chairperson
Dr. Susantha Herath
Dr. David Robinson

Abstract

Today, the amount of data generated is extremely large and is growing faster than computational speeds can keep up with. Therefore, using the traditional ways or we can say using a single machine to store or process data can no longer be beneficial and can take a huge amount of time. As a result, we need a different and better way to process data such as having data distributed over large computing clusters.

Hadoop is a framework that allows the distributed processing of large data sets. Hadoop is an open source application available under the Apache License. It is designed to scale up from a single server to thousands of machines, where each machine can perform computations locally and store them.

The literature indicates that processing Big Data in a reasonable time frame can be a challenging task. One of the most promising platforms is a concept of Exascale computing. This paper created a testbed based on recommendations for Big Data within the Exascale architecture. This testbed featured three nodes, Hadoop distributed file system. Data from Twitter logs was stored in both the Hadoop file system as well as a traditional MySQL database. The Hadoop file system consistently outperformed the MySQL database. The further research uses larger data sets and more complex queries to truly assess the capabilities of distributed file systems. This research also addresses optimizing the number of processing nodes and the intercommunication paths in the underlying infrastructure of the distributed file system.

HIVE.apache.org states that the Apache HIVE data warehouse software facilitates reading, writing, and managing large datasets residing in distributes storage using SQL. At the

end, there is an explanation of how to install and launch Hadoop and HIVE, how to configure the rules in a Hadoop ecosystem and the few use cases to check the performance.

Keywords

MapReduce, Computation, optimization, Java

Acknowledgement

I would first like to thank my starred Paper Advisor Dr. Dennis Guster of the Department of Information Systems at Saint Cloud State University. The door to Prof. Guster's Office was always open whenever I ran into a trouble spot or had a question about my research. He consistently allowed this paper to be my own work but steered me in the right direction whenever he thought I needed it.

I would also like to show my gratitude to Dr. Susantha Herath, Chair Department of Information Systems and Dr. David H. Robinson, Prof. Statistics Department for sharing their pearls of wisdom with me during this research. I am immensely grateful for their comments on an earlier version of the manuscript, although any errors are my own and should not tarnish the reputations of these esteemed persons.

Table of Contents

Page

List of Tables

List of Figures

Figure Page

Page

Page

Chapter 1: Introduction

Introduction

In this chapter, an overview of Hadoop and HIVE is explained. The nature and significance of the problem are discussed which will help in understanding the significance of Hadoop over traditional database MySQL when dealing with large datasets. The better understanding of different components of Hadoop ecosystem is explained.

Overview

Hadoop is the most popular open-source implementation of a single computing node or on clusters (Apache Hadoop, Wiki). Hadoop and MapReduce programs are used in dealing with a huge amount of data. Hadoop can be used for storing large data and for processing data such as data mining, report generation, file analysis, web indexing, and bioinformatics research.

As the name implies "Big Data" presents several challenges to Information System professionals. Most DBMSs are designed for efficient transaction processing: adding, updating, searching for, and retrieving small amounts of information in a large database.

It appears that platforms have been created to deal with the mass and structure of Big Data. Further, as one might expect they utilize distributed processing as well as software optimization techniques. An excellent summary of this work is presented by Singh and Reddy, 2014. In this work, they discuss both horizontal distributed file systems such as Hadoop (and its successor Spark) and vertical systems that rely on high-performance solutions which leverage multiple cores. This paper will focus on horizontal system Hadoop. The Hadoop file system and its associated components create a complex, but efficient architecture that can be used to support Big Data analysis.

In addition, this work explains in detail all the components in the Hadoop architecture including the map-reduce optimization software. Of special interest to this paper is the explanation of HIVE which is a MapReduce wrapper developed by Facebook, Thusoo et al., 2009. This wrapper provides a more efficient development environment due to its macro nature and makes the coding easier because programmers don't need to directly address the complexities of MapReduce code.

In sum, this paper will use a Hadoop-based data analytics ecosystem to support a Big Data application and compare its performance with a traditional DBMS. Special attention will be paid to the MapReduce function and the programming strategy associated with each solution.

Big Data

We are living in the data world. How to store and process data is the question. The data which is beyond the storage capacity and beyond processing power is called Big Data. Big Data is generated with different platforms and devices, for example, Facebook, sensors, networks, online shopping's, airlines, hospital data etc. These are data generated factors. In the year 1990's hard disk capacity was 130MB and gradually increased to 240MB and RAM was 65-120 MB approximately (Schmid, 2006). In 2017, hard disk capacity is minimum 500 GB -1 TB, RAM 4-16 GB. Hence, the data needs to be stored and not discarded. As a result, Hadoop has been introduced as the best solution for Big Data.

History of Hadoop

In 1990's Google had to come up with more data and to get the proper solution it has taken 13 years. In 2003, they had introduced GFS (Google File System) which is a technique to store huge data. In 2004, they have introduced MapReduce which is the best processing technique. They have published a "white paper" which has a description of GFS and MapReduce. Later, Yahoo which is the next best search engine after Google introduced HDFS in the year 2006-2007 and MapReduce was introduced in 2007-2008. They have taken the white paper which was given by Google and started implementing and came up with HDFS (Hadoop Distributed File System) and MapReduce. These are the two core components of Hadoop. Hadoop was then introduced by Doug Cutting in 2005.

Components of Hadoop

Figure 1 shows the components of Hadoop which are HDFS and MapReduce. Hadoop is an open source framework given by apache software foundation for storing and processing huge data sets with the cluster of commodity hardware which is done by these components.

HDFS. HDFS is a specially designed File System for storing huge data sets with cluster of commodity hardware with streaming access pattern which means "Write Once Read Many Times". The block size of each file is 64MB or 128MB. Shvachko, Kuang, and Radia (2010) stated that "in a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow in demand while remaining at every size."

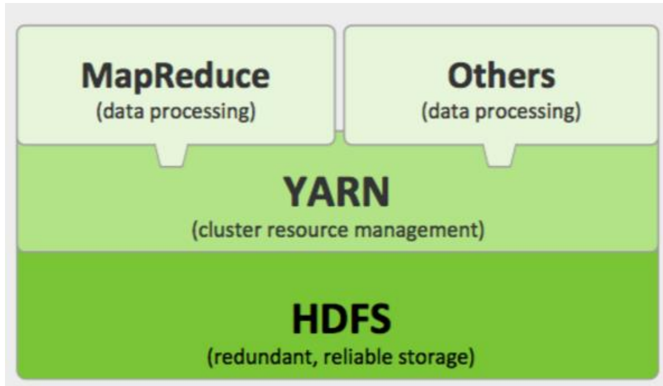


Figure 1. Components of Hadoop

Working of HDFS

- Suppose a client is willing to put 150MB of data in a cluster and sends a request to the NameNode cluster as metadata. Metadata stores the data about the data given by the Client.
- 150MB of data is stored in a file with the file name as file.txt as shown in Figure2.
- The file is divided into 3 input splits a.txt, b.txt, c.txt of each 64MB block size (150MB / 64MB).

a.txt – 64MB
b.txt – 64MB
c.txt – 22MB

Figure 2. File.txt input splits

- NameNode responds to the client and requests to store 150MB data in the nodes which has space.
- Client store all the txt files in different DataNodes. However, all the files need not be in sequence order.

- ❖ DataNodes are commodity hardware which means if the system goes down the data doesn't lose since HDFS has been given 3 replications by default. Hence it has 2 more backup files for each text files stored in different DataNodes. Hence, the a.txt file occupies 450 MB (150 MB * 3) of files in the whole cluster because of the replication. The same way other text files are also allocated to DataNodes with their corresponding replications. All the DataNodes which are SlaveNodes for that NameNode give proper block report and heartbeat to the NameNode. This acknowledgment gives the information of the condition of the DataNodes. Block report shows the DataNodes are still allocated with some size of block and heartbeat gives the status of the nodes. This is how the data is stored in HDFS.

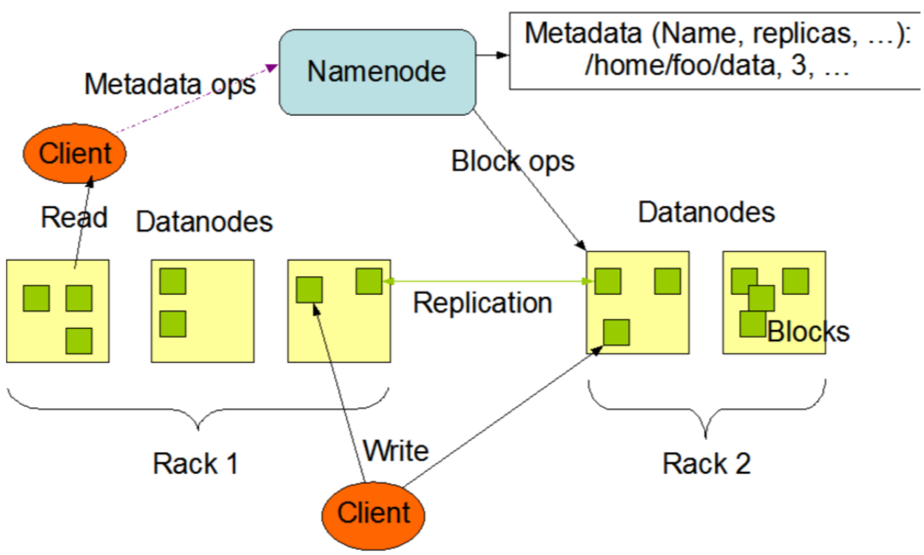


Figure 3. Procedure for storing Data in HDFS

- ❖ If the NameNode is lost then it's called "single point of failure" and nothing can be accessed in the system. (Borthakur, 2007)

❖ On the other hand, the JobTracker sends requests to NameNode when the client requests to process a file from those DataNodes. Then the NameNode checks for the file and sends the metadata to JobTracker. JobTracker assigns tasks to TaskTracker which process the files and gives the results to the client. Now the data is stored, the next step is processing of the stored data which is done by MapReduce.’

MapReduce. Counting number of occurrences of words is the basic concept in MapReduce. Suppose we have a file of 200MB which is divided into 4 splits of 64MB block size as shown in figure 4.

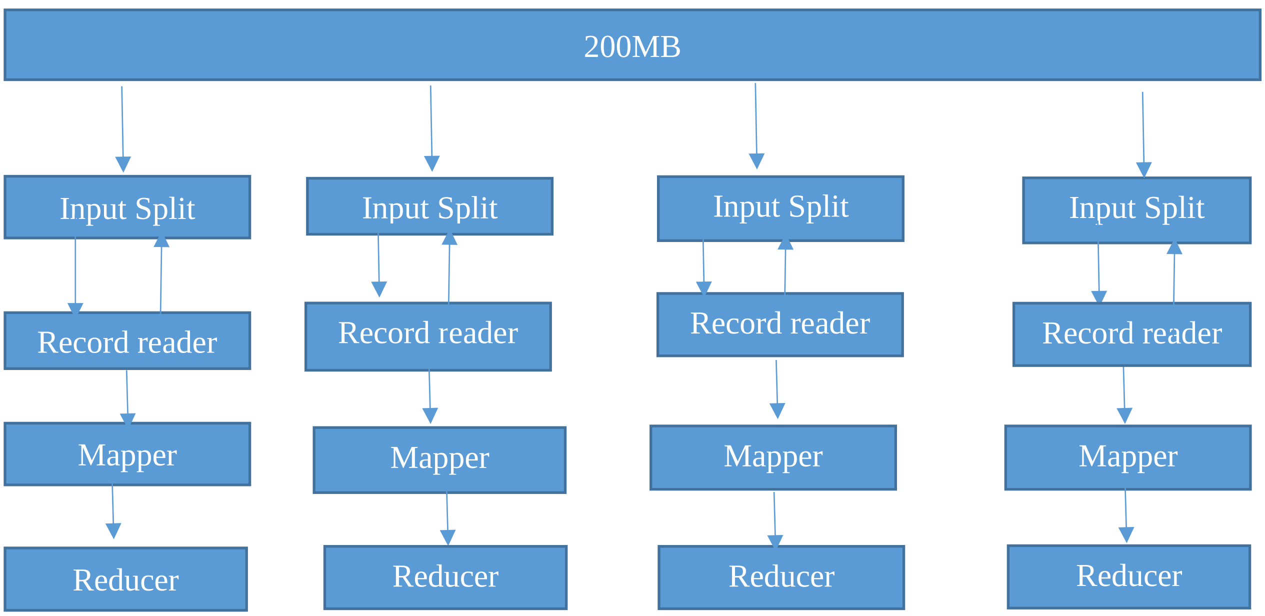


Figure 4. MapReduce process

Every input split has its own mapper. Hadoop can only run with MapReduce in the form of (key, value) pairs. Mappers and Reducers work with the (key, value) pairs. For every

Mapper/Reducer, a Record Reader is assigned which converts a text message in (key, value) pairs.

Record Reader is a predefined interface which converts the input file to (key, value) pairs. It takes the input file and converts the message in (byte offset, entire line) format where the byte offset is the address of the line. For instance, we have input splits as:

"Hi how are you" "How is your job"

The record reader takes the text input split and converts into (key, value) pair as follows:

Hi how are you - - - > *(0, hi how are you)*

Where 0 is, the byte offset and the text "hi how are you" is the entire line.

Next will be (16, how is your job). It counts the number of letters with the spaces between words from *"Hi how are you"* *"How is your job"*

Mappers run for every (key, value) pair. Since we are distributing the same job in multiple systems this concept is called parallel processing.

Problem statement

According to WorldWideWebSize.com, until April 2017, the web consists of approximately 4.5 billion web pages, a conservative number of web pages is approximately 30KB that translates to about a petabyte of data. The volume of the data generated every day is too fast and the traditional methods are not built to process this data in an organized meaningful manner. There is a need to conduct performance related research. Further, parallel processing is required to deal with huge data.

Therefore, this paper uses Hadoop with live data to test the performance of huge data after deploying in both traditional MySQL database and Hadoop.

Nature and Significance of the Problem

As the data is increasing with large velocity and in different forms, it is important to get the results quicker. Today's average disk speed reads about 120 MB/Sec (Michael, 2011). So, a single machine needs about three months to read the entire data and then MySQL Cluster is a famous clustered database that is used to store and manipulate data. "The problem with MySQL Cluster is that as the data grows larger, the time required to process the data increases and additional resources may be needed. With Hadoop and HIVE, processing time can be faster than MySQL Cluster" (Fuad, Erwin, & lpung, 2014). In this study, two data testers with the same data will run simple queries to compare the performance results.

Objective of the Research

The objective of this study is to compare the performance results of the traditional MySQL database with HIVE when dealing with huge amount of data. A recent study by Pol (2016) concluded that the drawback to using HIVE is that Hadoop developers must compromise on optimizing the queries as it depends on the HIVE optimizer and Hadoop developers need to train the HIVE optimizer on efficient optimization of queries. HIVE is generally used for processing structured data in the form of tables. In one article, it is explained that HIVE eliminates tricky coding and lots of boilerplate that would otherwise be an overhead if they were following MapReduce coding approach Refer dezyre.com for more details.

Research Questions and/or Hypotheses

Some of the questions that could be think about as going through this study are as follows

❖ Does HIVE provide better performance than MySQL when dealing with large amount of data?

- Yes, HIVE provides better performance for large data sets. However, MySQL still performs better results when dealing with small data sets. This can be seen in this study below.

❖ Does HIVE provide faster results with multiple DataNodes?

- HIVE provides faster results with multiple DataNodes. The study done by Thusoo et al. (2009) provides an excellent study on this question. It states that HIVE includes a system catalog- Metastore- that contains schemas and statistics, which are useful in data exploration, query optimization, and compilation.

❖ What was the role of MapReduce in this study?

- MapReduce feature is highlighted in the study for efficient, scalable processing of data by distributing the data in different DataNodes and doing the processing in parallel.

❖ How much time difference was between MySQL and HIVE when dealing with large data and small data?

- Refer Figure 44 and 45.

❖ Will block size of DataNodes be occupied or used by another task or is it left empty once the small amount of task less than 64MB is placed in it?

- As explained above in the working of HDFS the block size is occupied and used by another task and is not left empty.

Limitations of the Study

- Apache HIVE is very like MySQL and knows SQL clauses like FROM, WHERE, GROUP BY, ORDER BY. The drawback to using HIVE is useful only when the data is structured. With the unstructured, it is not a good tool. However, MapReduce can work on any type of datasets. In a study Kumar, Gupta, Charu, Bansal and Yadav (2014) explains some of the limitations of HIVE such as HIVE doesn't support for UPDATE & DELETE. It does not support singleton INSERT. Moreover, the study of (Rao, Sridevi, Reddy, & Reddy, 2012) found that Hadoop lacks performance in heterogeneous clusters where nodes have different computing capacity.

Definition of Terms

Table 1

Definition of terms.

Hadoop	Framework for distributed storage and processing of huge data
MapReduce	A programming model for large scale data processing.
HIVE	Used for performing queries
YARN	Used for scheduling jobs
DBMS	Database management system
HDFS	Hadoop Distributed File System
JSP	Java Server Pages

Summary

This study presents the processing time of HIVE and MySQL cluster on a simple data model with simple queries while the data is growing. Chapter 2 discusses the performance issues; background and literature review about Big Data and discusses the advantages of using Hadoop. Chapter 3 explains the methodology used in doing this research and provides the timeline and the future work which is to be done.

Chapter 2: Background and Literature Review

Introduction

This section provides the importance and need of using Hadoop. Also, the challenges of working with Big Data and briefly examined the architecture and performance issues related to the study. Further, research papers related to Big Data has also been discussed to better state the importance of Hadoop when compared to traditional database.

Challenges of working with Big Data

The literature indicates that there are many challenges when working in Big Data. Jagadish, Gehrke, Labrinidis, Papakonstantinou, Patel, Ramakrishnan, and Shahabi (2014) state that working in Big Data is a multi-step process and it is important not to ignore any of the steps. Specifically, they have identified the following required steps: acquisition, information extraction, data cleansing, data integration, modeling/analysis, interpretation, and reporting. Too often one or more of the steps are ignored and too much focus is placed on the reporting phase and the "visualization of the results" which often can result in erroneous reporting.

According to Fan, Han, and Liu (2014) "the massive size of big data leads to different challenges such as unique computational and statistical challenge, scalability, noise accumulation etc. Different areas of studies including the field of genomics, neuroscience, economics and finance face many challenges due to the high volume of data generated every day. This developing more adaptive and robust procedures". Hence, big data has drawn massive attention from researchers in Information Technology and other areas.

With Big Data, it is crucial to be able to scale up and down on-demand. Many organizations fail to consider how easily the Big Data project can grow and evolve. Constantly pausing a project to add additional resources will cut into times for data analytics. Refer *https://www.qubole.com/resources/solution/big-data-challenges/* for more Information.

Need for Distributed File Systems

As one would expect the increased volume of data that results from a Big Data concept complicates analytic endeavors. Because HDFS typically used for low-commodity hardware which means organizations need not spend a lot of money on purchasing hardware of high quality. Distributing data into multiple machines not only saves time but makes the job easier to process. In this study, HDFS is used for performing distributed processing of data which is a Hadoop-based component. If helps in ETL process and gives the processing result of large data in seconds. In an Independent study performed with Punith Etikala (Sultana & Etikala ,2015) we found that when we are dealing with traditional database with relatively large amount of information MySQL system was crashed. However, Hadoop with the help of distributed file system performed analysis in a few minutes. Levy and Silberschatz (1990) stated that the purpose of distributed file system is to allow the computers share the same data and resources by using a common file system. It can also provide high – throughput and suitable for applications with large data sets.

Distributed file systems also help multiple users on different machines to share files in the share resources. It differs in many ways from traditional database systems. Some of the differences are performance, handling of nodes, handling of temporary or permanent loss of data storage or resources.

Distributed storage is relatively very easy to understand when compared to traditional way of storing the data. However, it has complex configurations and management. According to

(Microsoft, wiki) “DFS provides location transparency (via namespace content) and redundancy (via the file replication component) to improve data availability in the face of failure or heavy load by allowing shares in multiple different locations to be logically grouped under one folder, or DFS root.”

Also, as we discussed earlier the traditional MySQL requires high RAM and disk space but the work in Distributed file system is done in parallel.

Architectures to Support Big Data

There appears to be a consensus that the concept of a distributed file system offers an excellent platform to support Big Data. While there may be other viable options in terms of design or functionality, but distributed file systems by far offer the most cost effective solution (Jarr, 2014). A prime example of this is Hadoop, which is designed to deploy a distributed file system on cheap commodity machines (Reed & Dongarra, 2015).

It also is interesting to note that the architecture to capture the Big Data in the first place is expanding as well. This environment is personified by the Internet of Things (IoT) concept. IoT relies on interconnected physical objects which effectively creates a mesh of sensor devices capable of producing a mass of stored information. These sensor-based networks pervade our environment (e.g., cars, buildings, and smartphones) and continuously collect data about our lives (Cecchinel, Jimenez, Mosser & Riveill, 2014). Thus, the use of IoT will further propagate the legacy of Big Data.

Performance Issues with Big Data

Big data is defined as a large amount of data which needs to be processed by using different technologies and architectures. It is expected to have performance issues when working with big data**.** However, Big Data due to its various properties results in many challenges. Jewell et al. (2014) have identified four dimensions:

1. volume (Big Data applications must manage and process large amounts of data),
2. velocity (Big Data applications must process data that is arriving more rapidly),
3. variety (Big Data applications must process many kinds of data, both structured and Unstructured) and
4. Veracity (Big Data applications must include a mechanism to assess the correctness of the large amount data of rapidly).

It eliminates the need of extensive expensive hardware and storage space. When the data is stored in different modules like a public cloud, private cloud, Cloud computing needs to be introduced which the most powerful technology to perform complex is computing on the datasets. In the study Hashem, Yaqoob, Anuar, Mokhtar, Gani, and Khan (2014) addressed the issues on the rise of big data in cloud computing and explained “Addressing big data is a challenging and time-demanding task that requires a large computational infrastructure to ensure successful data processing and analysis”.

Jacobs (2009) pointed out that “just as maintaining locality of reference via sequential access is crucial to processes that rely on disk I/O (because disk seeks are expensive), so too, in distributed analysis, processing must include a significant component that is local in the data—that is, does not require simultaneous processing of many disparate parts of the dataset because communication between the different processing domains is expensive)". Just as it is easy to extract the data from the systems it should be easy to store the data as well. So, one can understand that it is easy to get the data in parallel processing instead of storing in databases and performing analysis with the traditional databases. Surely, the whole concept of distributed parallel processing of data seems to be easier but it has lots of limitations including the management of the file system. Hence the future systems and configurations need to enhance more beyond the present state.

Jacobs (2009) also stated that the business applications, at least, a data warehouse is regarded as the solution for the database problems. The normal way used in this data ware-housing is extracting the data from one database and transferring and loading the data in another database for performing queries to get the analysis which is so-called ETL process. To understand ways to avoid the pathologies of big data in any context it is important to consider what makes it big. Hence how big the data is it is more difficult to maintain multiple copies of the data.

Advantages of Hadoop and MapReduce

On a basic level, the advantage of Hadoop is that it provides an efficient and cost-effective platform for distributed data stores. MapReduce then provides the means to connect the distributed data segments in a meaningful way. MapReduce with Hadoop helps analyze data in more efficient and timely manner. In that sense, MapReduce can also be used with parallel DBMS. Along with this Hadoop offers support of multiple languages that is used for processing and storing of data. Time manner is used to connect in a distributed structure. A pertinent research project utilized an open-source MapReduce implementation in conjunction with two parallel DBMSs, (Stonebraker, Abadi, DeWitt, Madden, Paulson, Pavlo, & Rasin, 2010). They determined that DBMSs are much faster than MapReduce open source systems at the point that data is loaded. However, loading the data requires much longer to load in the database systems. Dean and Ghemawat (2010) clarified the inter-relationship between MapReduce and parallel databases. Specifically, they determined that MapReduce provides many significant advantages over parallel databases. First and most important, MapReduce introduces fine-grain fault tolerance within large jobs. This check-pointing logic allows for easier recovery when a failure occurs in the middle of a multi-hour job. Second, MapReduce is more versatile in facilitating data processing and data loading in a heterogeneous system containing different storage architectures. Third, MapReduce provides an excellent schema in managing the execution of complex functions which are not directly supported by the SQL language. Last, MapReduce besides having performance advantages provides an effective way of linking complex data parts together within any architecture but shines when linked with Hadoop (Reed & Dongarra, 2015).

Architecture and Performance Issues

It has been established that the volume of processing within Big Data requires a well-designed architecture if reasonable performance is to be obtained. The volume of processing within Big Data necessitates a well-designed architecture if reasonable performance is to be realized. As stated earlier a study by of Reed and Dongarra (2015) provides an excellent overview of Exascale computing. The prime feature of this architecture revolves around a distributed storage system which permits the data to be extracted from multiple devices simultaneously (Chang et al., 2008). As one would expect the Hadoop file system adheres to this logic. A cost benefit of Hadoop is that it can be considered a data-analytics cluster based on commodity Ethernet networking technology and numerous PC nodes (even a generation or two old) containing local storage. This design had received excellent reviews in providing a cost-effective solution for large scale data analytics (Lucas et al., 2014). Given this architecture on could then view Hadoop as the logic to bind the components together. This situation facilitated the creation of a test-bed environment for this paper. The fact that cloud computing was being used made the resources available to rapidly configure it in the author's private cloud using virtualization software.

A prime part of the Hadoop system implementation strategy is the Map Reduce model (Dean & Ghemawat, 2004). Specifically, Map Reduce is designed to support the parallel processing function within Hadoop applications. To fit well in cloud computing it is designed to utilize multi-core as well as processors distributed across multiple computing nodes. Of course, the foundation of the Map Reduce system is a distributed file system. Its major function is based on the simple concept: Large files are reorganized into equal size blocks, which are then distributed across a cluster and stored. In this paper, the storage occurred within a private cloud. To ensure reliability fault tolerance was implemented which means that each block is stored several times (at least three times) across computers nodes.

A challenge with undertaking a performance analysis of this type is dealing with new technology and learning new things. The authors' primary background in dealing with large data sources was a traditional relational database structure. Fortunately, a couple of tools are available to assist in extracting data from the Hadoop file system. First, there is "PIG" which was devised by Yahoo! to streamline the process of analyzing large data sets by reducing the time required to write mapper and reducer programs. According to IBM 2015B, the pig analogy stems from actual pigs, who eat almost anything, hence, the PIG programming language is designed to handle any kind of data! While it boasts a powerful programming language it is basically new syntax and requires time to master. Another option HIVE uses an SQL derivative called HIVE Query Language (HQL) so that the developer is not starting from scratch and has a much shorter learning curve. While HQL does not have the full capabilities of SQL it is still useful (IBM, 2015A). It completes its primary purpose quite well which is to serve as a front end to simplify MapReduce jobs that are executed across a Hadoop cluster.

Because the goal of this paper is to assess the performance using similar data sets stored under two different structures it is important to be able to transfer the exact data between two different data structures. A tool called SQOOP was used to solve this problem.

According to (Sqoop.apache.org), SQOOP is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. Its use is critical for this project because the original data being utilized is stored in a MySQL database. Specifically, this data came from the twitter logs and consisted approximately 400,000 records. By using Java and MySQL, the number of records of TweetData table is regenerated. Currently, the number of records are approximately 250 million.

Summary

The study of different papers related to the work of this paper is discussed and some important issues and challenges of Hadoop have been discussed in detail. The next chapter gives the brief description of the methodology used in this study and the hardware and software environment required for proceeding forward in the work.

Chapter 3: Methodology

Introduction

This section provides the design and architecture of Hadoop, the detailed information of what tools have been used and the total number of data collected from different resources. The data used in this paper is in a structured format and contains millions of records which are in the form of HIVE table and in MySQL tables. Hadoop runs with the help of different tools and techniques which are also discussed in detailed. To perform the analysis the hardware and software requirements are required, the amount of space and memory required for each of the Virtual Machines is collected.

Design of the Study

The following diagram explains the architecture of Hadoop used for this project. “It is centered on the HDFS file system which is used to store the data. To achieve the desired parallelism MapReduce and YARN framework is used to process HDFS data and provide resource management. Apache HIVE is built on top of Hadoop to provide a data summarization and analysis on HDFS data. Apache SQOOP is used to transfer data between relational databases and the HDFS system. Finally, when the data is stored in both MySQL and HIVE databases analysis is performed on the data and the results are compared.” (Etikala, 2016).

A drawing that depicts the process that was followed to undertake the experimental comparison appears below. Both the MySQL database and the HDFS were run on similar hardware within the same cloud. However, the HDFS system was distributed across several nodes. As would be expected the HDFS system performed better in all the experimental trials.

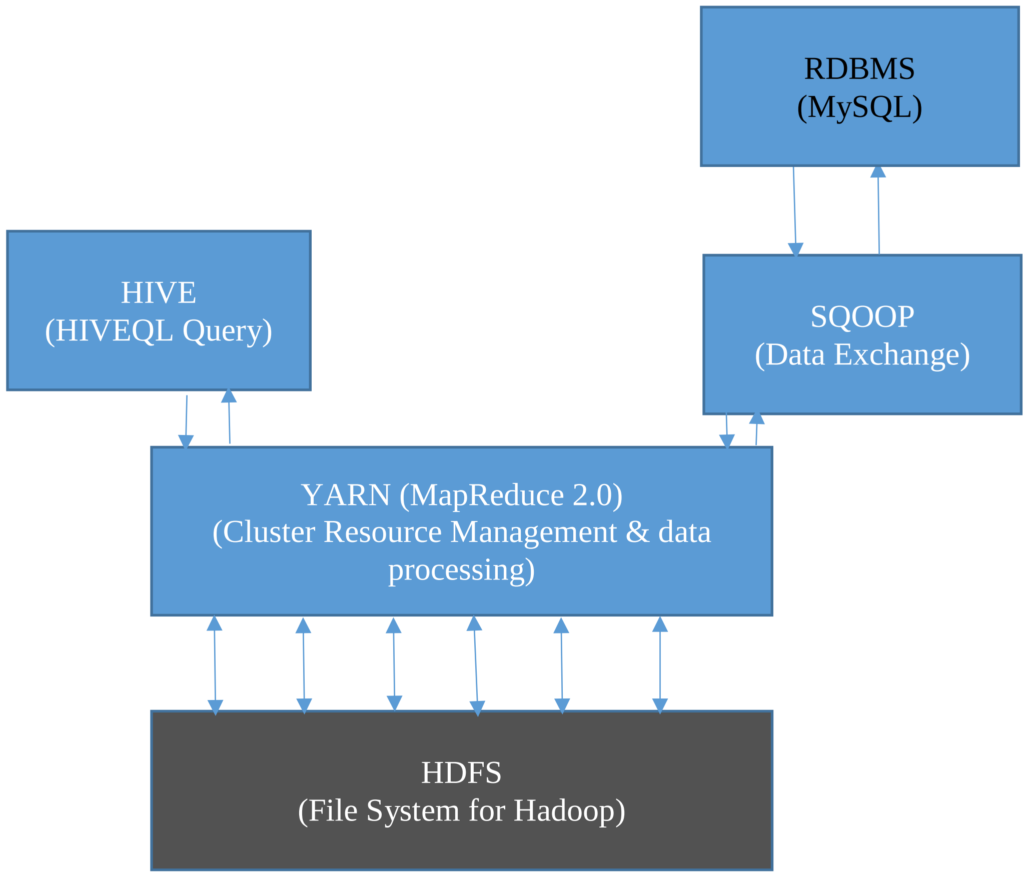


Figure 5. Architecture Diagram

The main idea of the project is to compare the performance of MySQL and HIVE and to prove that with large data sets, HIVE gives better performance that the traditional database (MySQL). The architecture diagram explains the process which will be followed in this project. The huge data collected Twitter is transformed into MYSQL in a structured format. Using SQOOP, which is a data transfer tool, the data from MySQL is loaded into Hadoop distributed file system. The data is then moved to HIVE in the form of structured tables.

Using HIVE Query Language (HQL), some queries are performed on the data and on the other side, using MYSQL the same queries are performed.

Yarn framework is used for job scheduling and cluster resource management. MapReduce framework is used to split data into small pieces and execute the related jobs on nodes. The results will be collected from nodes, integrated and then return to users. In this way, MapReduce transforms a single-node processing job to a parallel processing job to improve the execution efficiency.

Data Collection

The data used in this project is taken from Twitter App for the performance analysis. The number of records is approximately 250 million. By using Java and MySQL, the number of records of TweetsData table is regenerated.

This allows downloading real-time data available from Twitter company server. The website "https://apps.twitter.com/" allows creating a Twitter App.

In the Application Management window, "Create New App" allows to create an application.

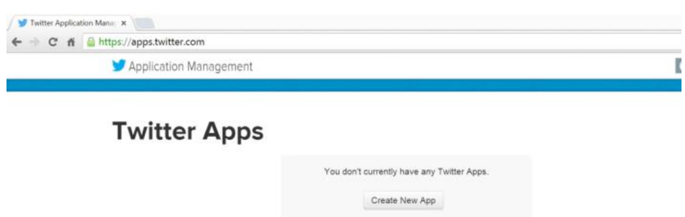


Figure 6. Twitter Application Management

Once the application is created successfully, In the Application Management screen, the newly created Twitter App appears.

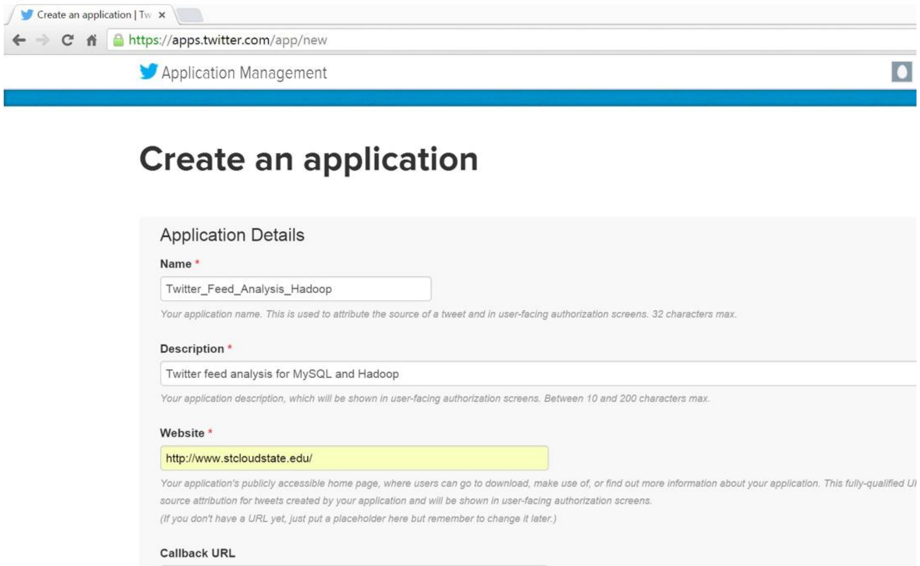


Figure 7. The template of creating an application

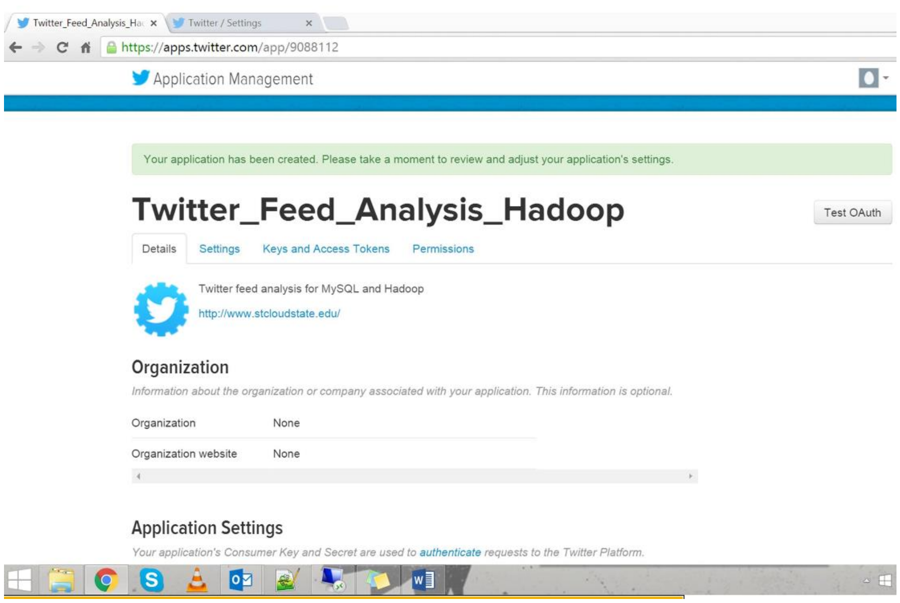


Figure 8. Twitter Application

Open the newly created Twitter Application, and navigate to “Keys and Access Tokens” tab, where Consumer Key (API Key), Consumer Secret (API Secret), Access Token and Access

Token Secret are the 4 secret keys, which allows Java program to connect to Twitter App to retrieve the data from Twitter company server.

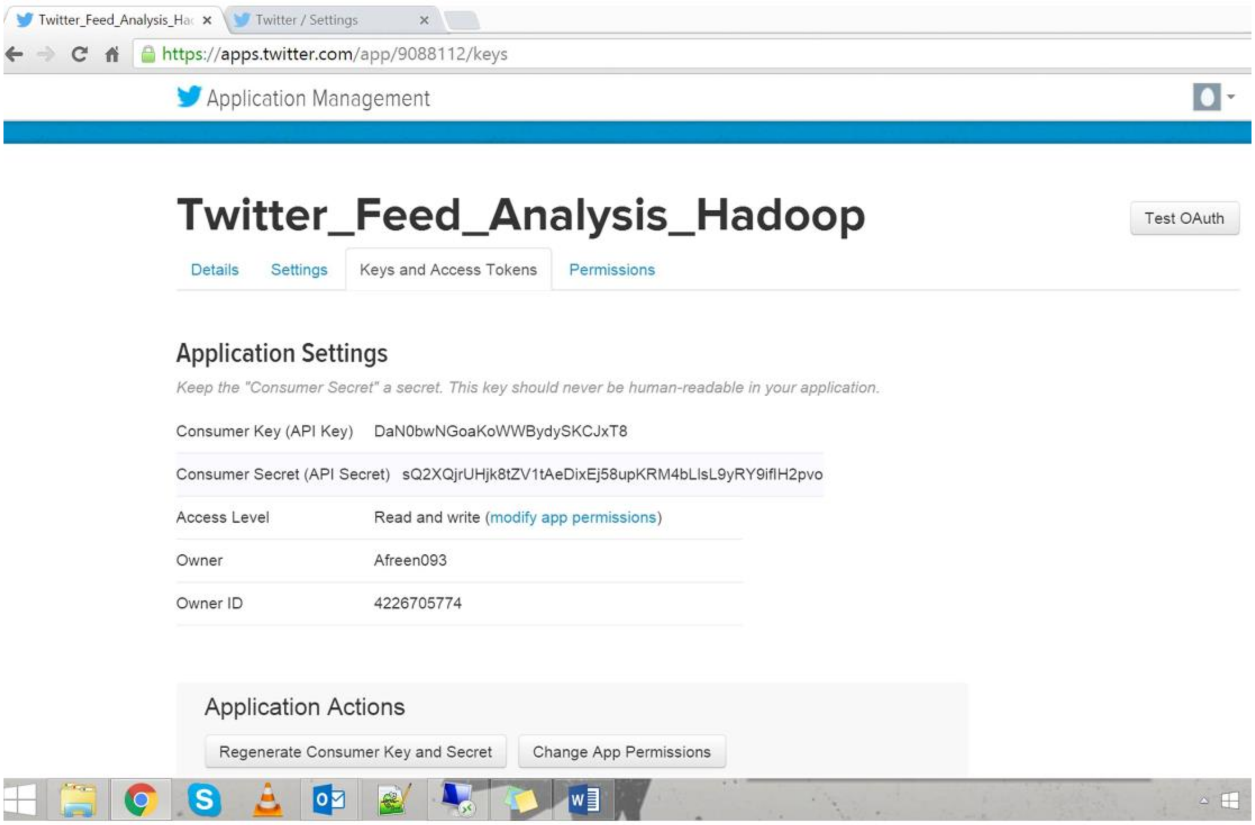


Figure 9. Twitter Application Key and Access Tokens Management

To perform analysis with Hadoop, 20GB of data gathered from the Twitter server. TwitterData.java is used to download raw data, which is in JSON format. Converter.java is used to parse the JSON data and gather the required data to perform analysis with Hadoop. There are four main “objects” that will be encountered in the API: Tweets, Users, and Entities (see also Entities in Objects), and Places in the feeds. A similar study is done by (Etikala, 2016).

Tools and Techniques

Hadoop is the most popular platform for Big Data analysis. It is huge and involves many supporting frameworks and tools to effectively run and manage it. Since Hadoop runs on Java, there are some required pre-requisites that need to start Hadoop. Below are the tools and techniques used in this project are SQOOP, HIVE, MySQL, MAPREDUCE and Hadoop Daemons.

HIVE. It is a data warehouse built on top of Hadoop and is used for analyzing, summarizing and querying of data using HIVE Query Language (HQL) (Apache Hadoop, Wiki). These queries are compiled into map-reduce jobs which are executed by Hadoop. “HIVE was open sourced in August 2008 and since then has been used and explored by several Hadoop users for their data processing needs (Thusoo et al., 2009). Generally, HIVE runs on our workstations and converts SQL queries into a series of MapReduce jobs for execution on Hadoop cluster (White, 2012). The data is organized in the form of tables using HIVE”. It is the module that allows the extraction logic of the data to be formulated using an SQL-like language.

SQOOP. It is a tool to transfer data from one database to the other. In this paper, SQOOP is mainly used to transfer data from MySQL to HIVE. It splits each table into four parts by default and it uses the mapper of MapReduce framework to store data in clusters via JDBC driver during data migration (Sqoop User Guide, v1.4.5). Data from the tables is then stored in the Virtual Machines where Hadoop executes the Mappers randomly. The data is therefore distributed in the VM clusters. Microsoft uses SQOOP based connector to help transfer data from Microsoft SQL server database to Hadoop. SQOOP uses MySQL dump to fetch the data stored in MySQL.

MYSQL. It is the open source relational database management system and it is widely used in web applications. It is a central component of the widely-used LAMP open source application software. LAMP includes Linux, Apache, MySQL, and Perl/Python/PHP (MySQL, wiki). It is useful for managing MySQL database and managing data using various SQL statements such as INSERT, UPDATE, and REVOKE, SELECT, DELETE as well as JOINS.

It plays a very important role in many Big Data platforms, including those implemented by Facebook and Twitter. MySQL is beneficial to the developers because of its speed, reliability, data integrity and scalability. It can successfully process huge amounts of data (terabytes of data) but as the data increases, the time required to process the results increases as well and additional resources are required as well.

MAPREDUCE. Google invented MapReduce and it has been used to analyze the entire internet. Analyzing real weather data and E- Commerce data can also be performed (Fang, Sheng, Wen, & Pan, 2014). MapReduce is the heart of Hadoop. It is a programming model that allows large scalability across thousands of clusters. “The term MapReduce refers to two separate distinct tasks that Hadoop programs perform” (Quintero et al., 2015). The first is the map job, which takes input data and processes it to produce key/value pairs. The reduce jobs take the key/value pairs and then combines and aggregates them to produce a result. As the name MapReduce implies, the reduce job is always performed after the map job. It offers network load reduction and faster computation.

HADOOP DAEMONS:

According to the Apache Hadoop, Wiki "A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode, and DataNode."

Hadoop consists of five daemons. They are divided between the master node and SlaveNodes. Master daemons consist of three Hadoop daemons such as the NameNode, SecondaryNameNode and a JobTracker. Whereas, the slave daemons are the DataNodes and the TaskTracker. Daemon is a background process. Every master service can talk to each other and all slave daemons can interact with each other. If NameNode is a Masternode its corresponding SlaveNode is DataNode. JobTracker talk to TaskTracker. If the NameNode is JobTracker its corresponding SlaveNode is TaskTracker as shown in the figure below.

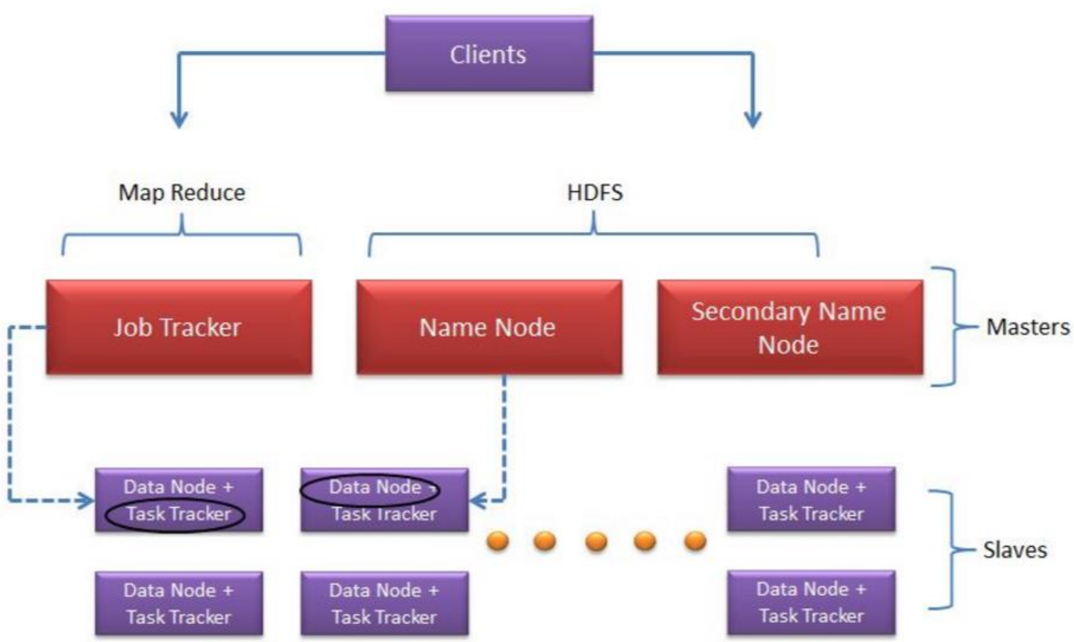


Figure 10. HDFS daemons and Hadoop Core Components

NameNode. It stores and maintains the metadata of HDFS and tracks where the data file is kept across the cluster. It is a single point failure for HDFS which means when the NameNode goes down, the file system goes offline.

Secondary NameNode: It is used to perform the housekeeping functions for the NameNode. It can be hosted on a separate machine and acts as a backup.

JobTracker. It manages the MapReduce jobs and distributes individual tasks to the machines running the TaskTracker.

DataNode. It stores actual HDFS data blocks

TaskTracker. It is mainly responsible for instantiating and monitoring individual map and reduce tasks. A heartbeat is sent from the TaskTracker to the JobTracker every few minutes to check its status (Apache Hadoop, wiki)

One important thing to keep in mind is that Hadoop master nodes don't talk to the SlaveNodes. However, all the DataNodes can talk amongst themselves. The metadata is stored in the namespace of the NameNode which keeps track of all the tasks that's being done.

Hardware and Software Environments

- ❖ 3 Virtual Machines with Ubuntu 14.04.3 Operating System

Table 2

Virtual Machine Details

IP Address	Number of Cores	RAM	CPU Clock Speed
10.31.10.102	8	25GB	2200Mz
10.31.10.103	2	4GB	2200Mz
10.31.10.104	2	4GB	2200Mz

- Java 1.6.0_40
- OpenSSH 6.6.1
- MySQL Server 5.5
- Apache Hadoop 2.6.2
- Apache HIVE 1.2.1
- Apache SQOOP 1.4.6 – For Hadoop 2.x

Summary

The main idea and concept of this project are discussed along with the architecture required for it. Data used in this project is discussed and the tools required to start Hadoop and its services are discussed in detail. Some of the core components of Hadoop are NameNode, DataNodes, Secondary NameNode, JobTracker, and TaskTracker. The next section gives the brief description of how the data is provided and analyzed to see the performance of the traditional database and HIVE database.

Chapter 4: Implementation

Introduction

This section provides the detailed information of how the data is presented and used for processing and analysis. The installation steps of the pre-requisites discussed above are shown clearly which helped in Hadoop setup.

Data Presentation

Below is the program listing and step by step procedure to execute it.

<u>Installation of Java:</u>

Follow the following commands to update package index and install Java Runtime Environment:

```
sudo apt-get update
sudo apt-get install openjdk-default-jre
```

The openjdk-default-jre package contains just the Java Runtime Environment. If you want to develop Java programs, then install the openjdk-default-jdk package.

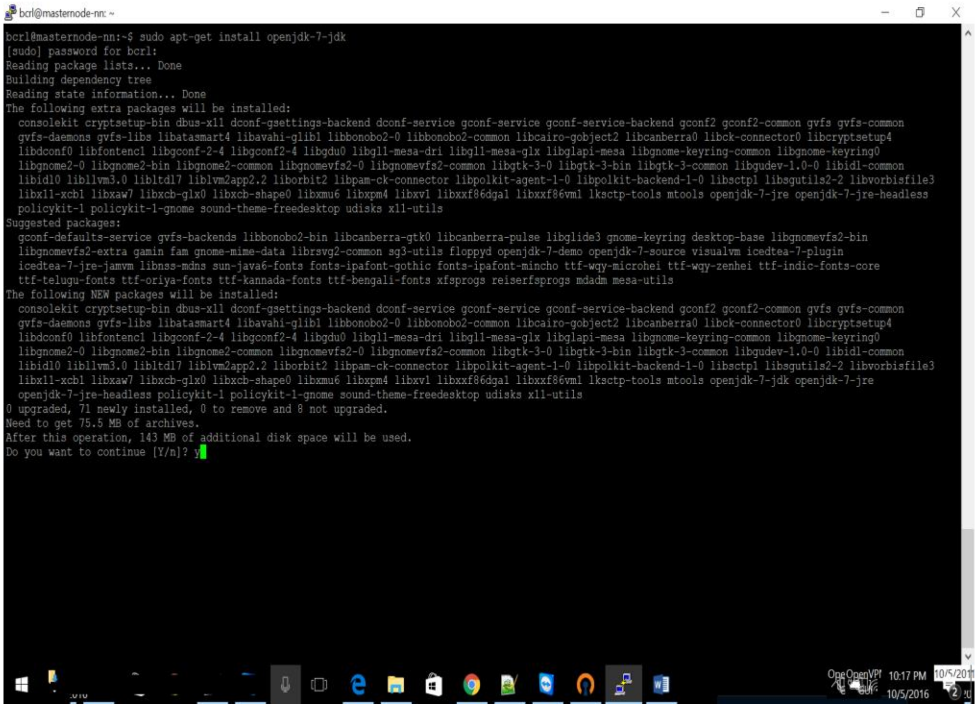


Figure 11. Java Installation

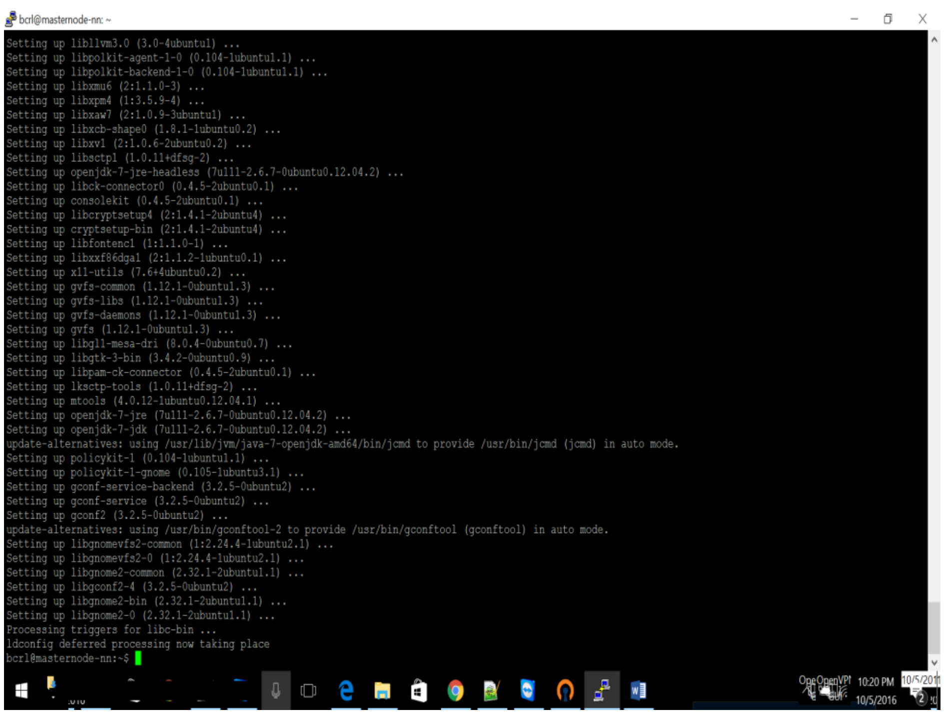


Figure 12. Extracting Java Jar files

The following command is used to verify that java installed.

java -version.

The project used Java version 1.6.0.

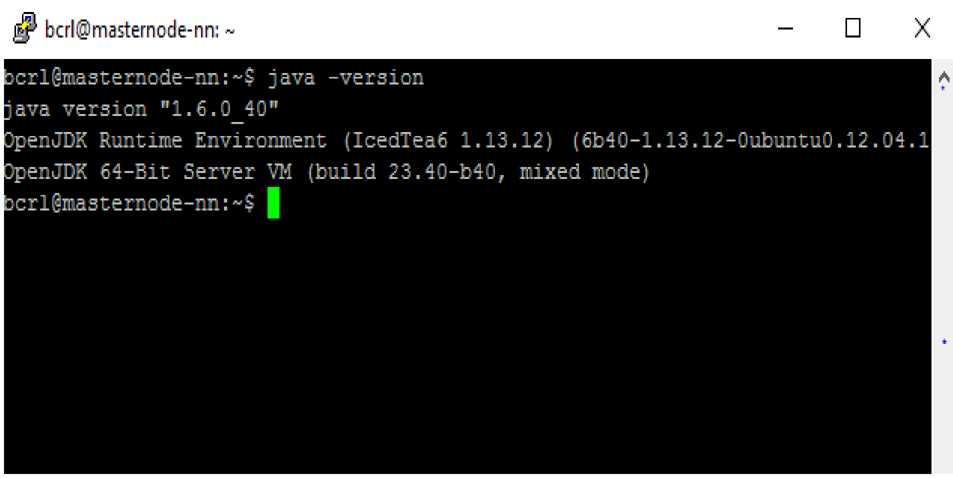


Figure 13. Java Version

Installing SSH:

There are two components of SSH:

SSH: This command is used to connect to remote client machines, generally done by the client.

SSHD: Daemon, which runs on the server, allows the clients to connect to the server.

Install SSH by using the following command.

sudo apt-get install ssh

To locate the pathname which would run if SSH or SSHD commands were executed. Installation of SSH can be verified by 'which' command.

which ssh

/usr/bin/ssh

which sshd

/usr/sbin/sshd

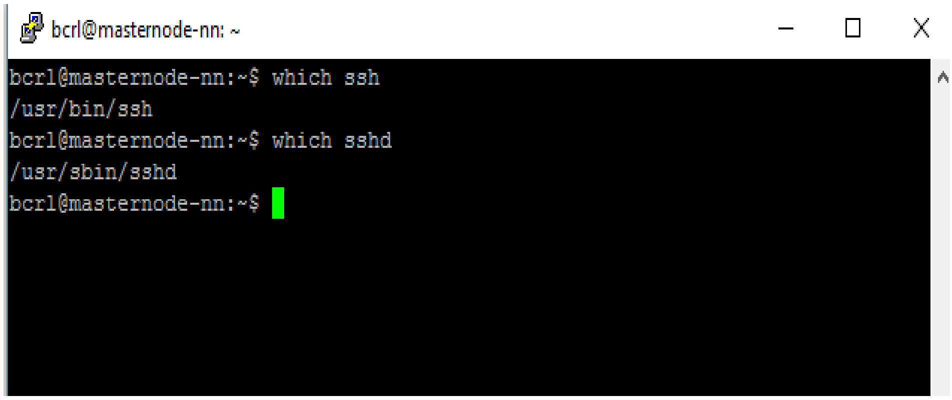


Figure 14. SSH and SSHD verification

MySQL Installation:

MySQL is a widely-deployed database management system used for organizing and retrieving data.

* Install MySQL server

To install MySQL, open terminal and type in these commands:

sudo apt-get install MySQL-server-5.5

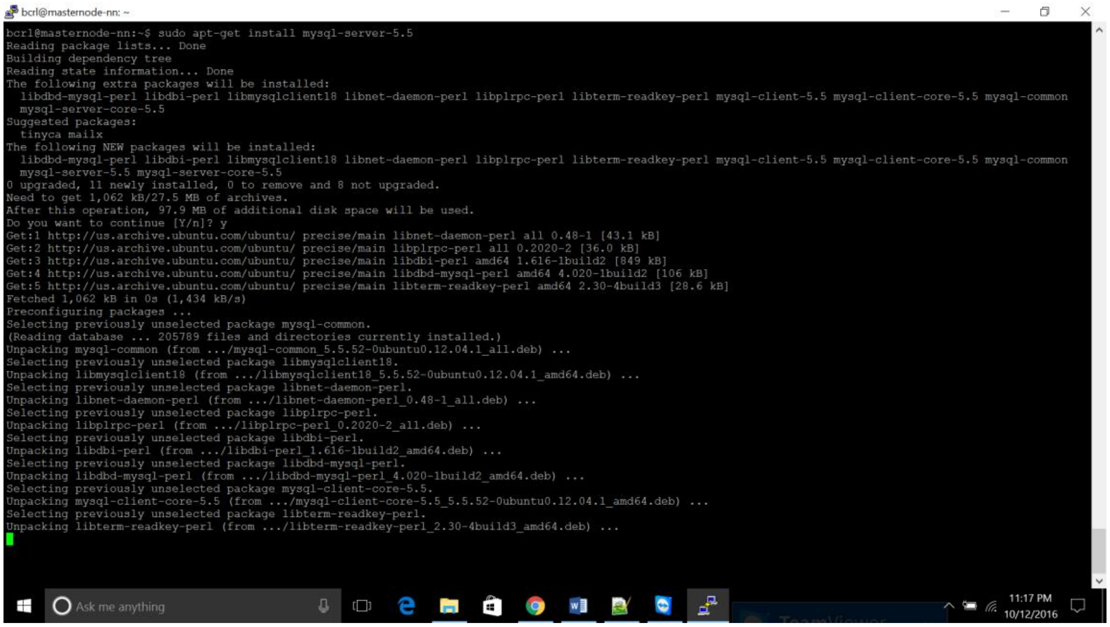


Figure 15. Installing MySQL

During the installation, MySQL will ask you to set a root password (new password & re-type password), which allows users to connect to MySQL as root.

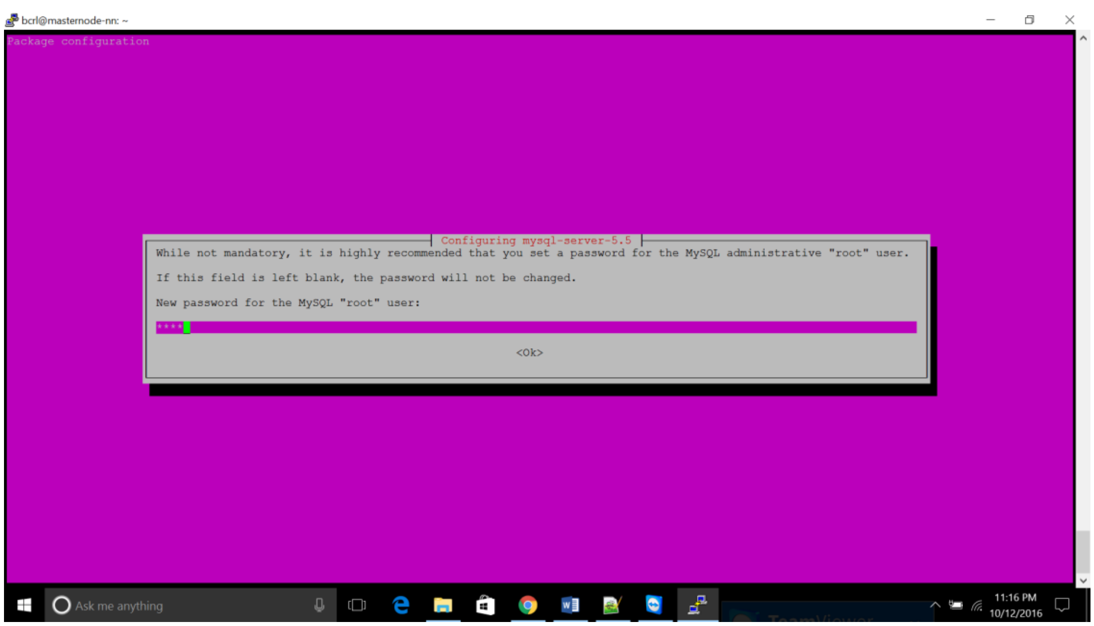


Figure 16. Assigning credentials for MySQL

- Know machine your IP Address

  hostname -

- Configuring Machine IP to MySQL

  sudo nano /etc/MySQL/my.cnf

  bind-address = 10.31.10.102

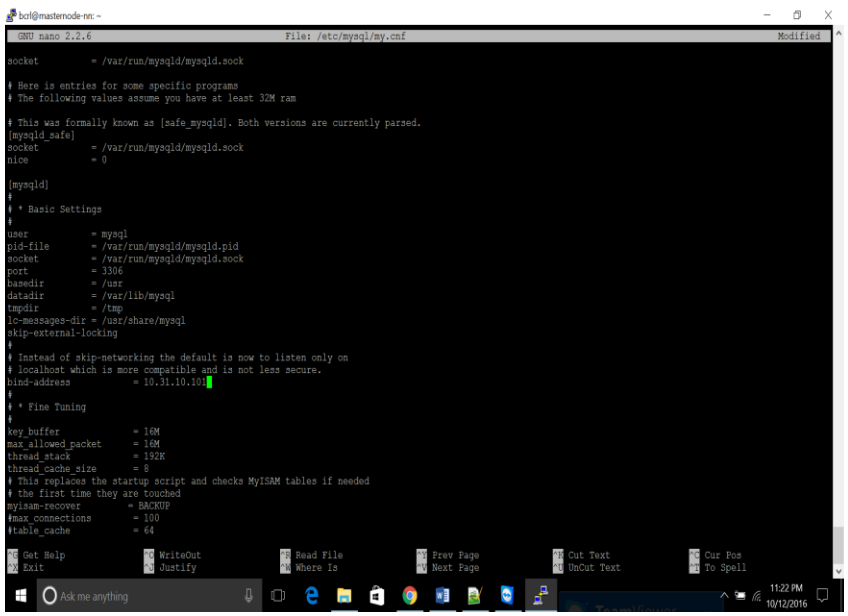


Figure 17. Configuring Machines IP address to MySQL

Restart MySQL service, which allows MySQL to use the configured IP address.

sudo /etc/init.d/MySQL restart

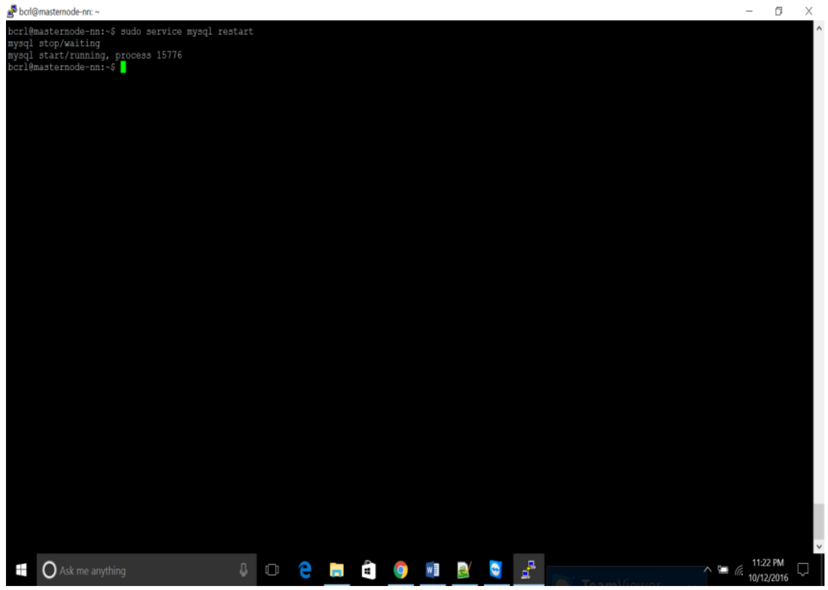


Figure 18. Restarting MySQL

- Verifying MySQL

MySQL-u root -p

Enter password: root

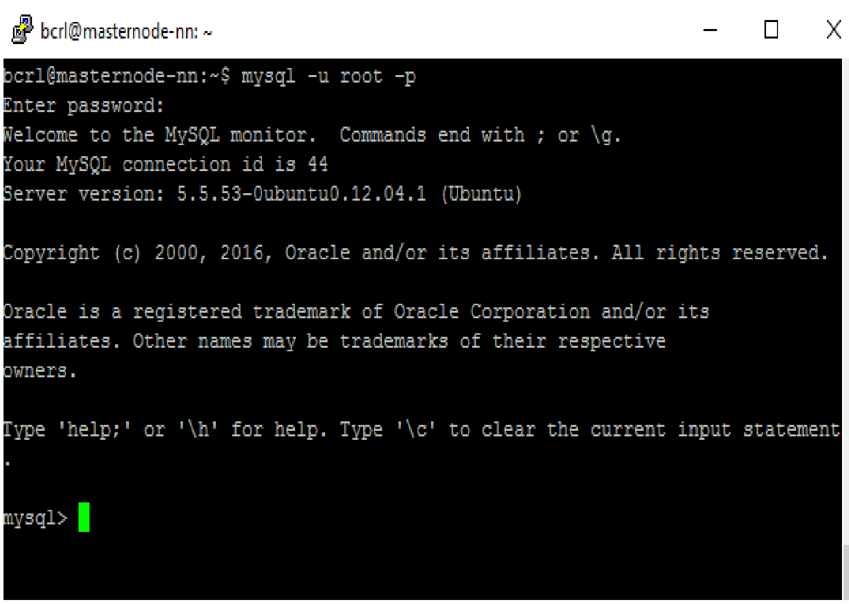


Figure 19. MySQLInstallation

MySQL> show databases;

MySQL> exit;

MSQL Installation is complete

SQOOP Installation:

SQOOP is mainly used to transport data from RDBMS to HDFS & HDFS to RDBMS.

- Downloading SQOOP

  Download SQOOP binary distribution by following command:

  wget http://download.nextag.com/apache/SQOOP/1.4.6/SQOOP-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz

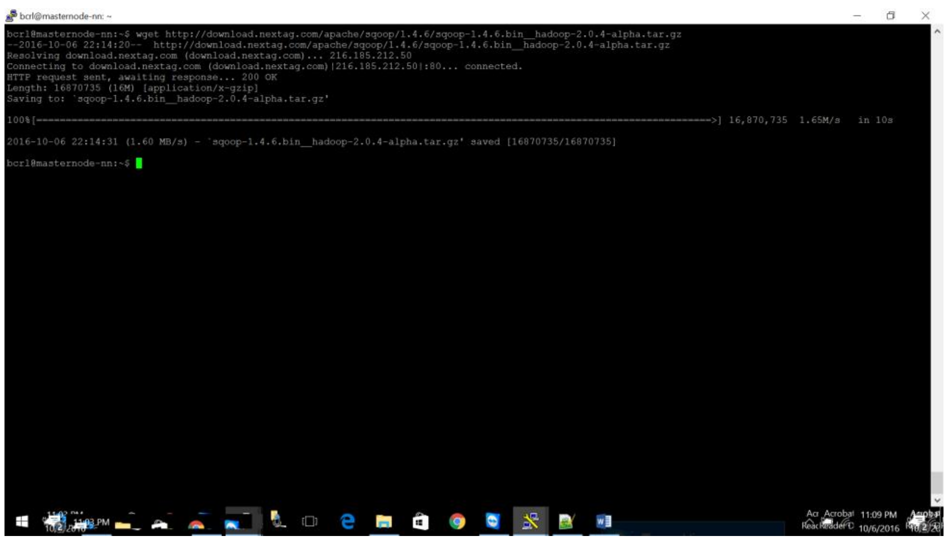


Figure 20. Downloading Installation

- Installing SQOOP

  The following commands are used to extract the SQOOP tar ball and move it to "/home/bcrl/SQOOP" directory.

  tar xvzf SQOOP-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz

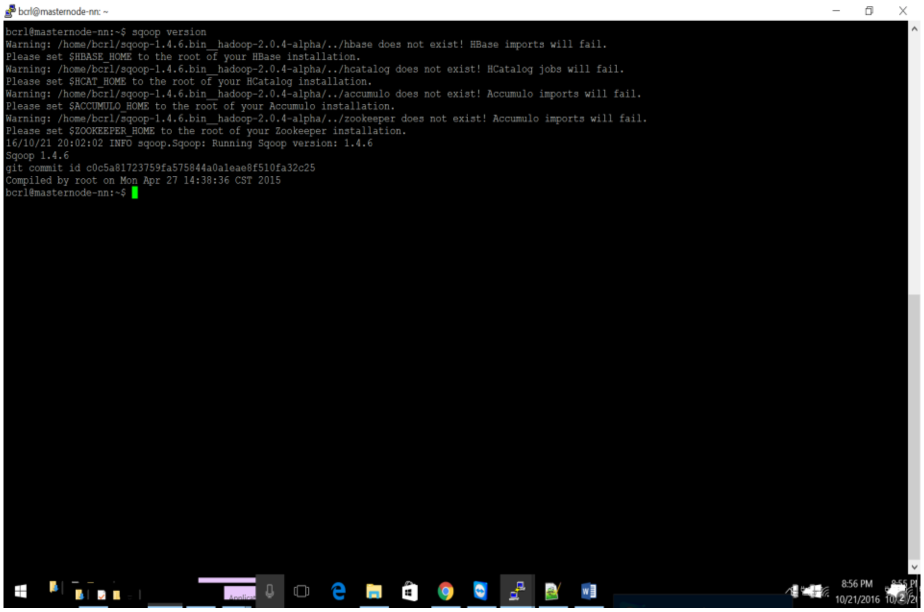


Figure 21. SQOOP Installation and Extraction

sudo mkdir SQOOP

cd SQOOP-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz/

sudo mv * /home/bcrl/SQOOP/.

- Changing owner and group for SQOOP installation directory to Hadoop dedicated user

  sudo chown -R bcrl:bcrl; /home/bcrl/SQOOP

- Configuring bashrc

  nano ~/. bashrc

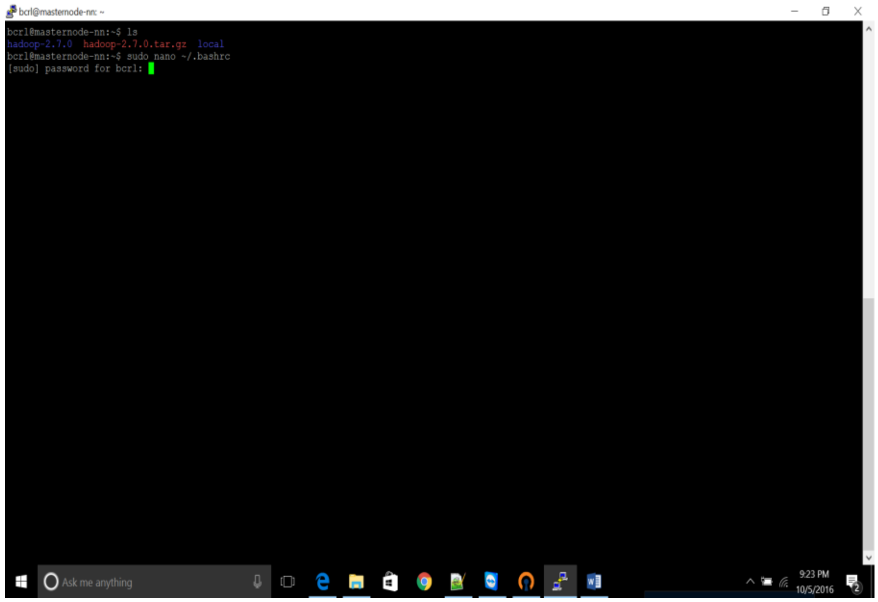


Figure 22. configuring bashrc for SQOOP

In bashrc file append the following statements:

#SQOOP VARIABLES START

export SQOOP_HOME=/home/bcrl/SQOOP

export PATH=$PATH: $SQOOP_HOME/bin

#SQOOP VARIABLES END

source ~/. bashrc

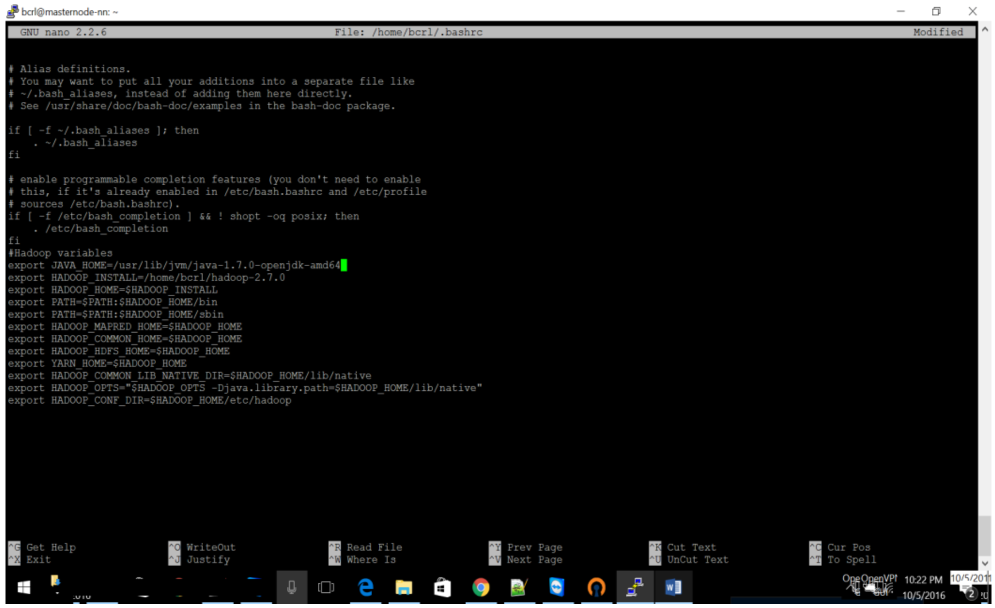


Figure 23. Appending commands for bashrc configurations

- Configuring SQOOP (optional if HADOOP_COMMON_HOME and HADOOP_MAPRED_HOME configured in hadoop-env.sh in Hadoop configurations directory)

  To configure SQOOP with Hadoop, you need to edit the SQOOP-env.sh file, which is placed in the $SQOOP_HOME/conf directory. First of all, Redirect to SQOOP config directory and copy the template file using the following command.

      cd $SQOOP_HOME/conf

      mv SQOOP-env-template.sh SQOOP-env.sh

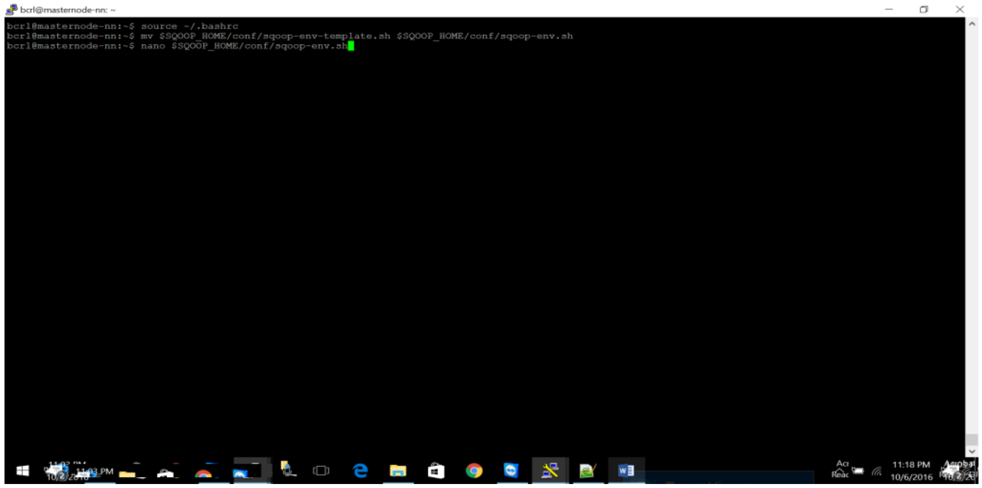


Figure 24. Redirecting to SQOOP configuration directory

Open SQOOP-env.sh and edit the following lines:

nano SQOOP-env.sh

export HADOOP_COMMON_HOME=/home/bcrl/hadoop-2.7.0

export HADOOP_MAPRED_HOME=/home/bcrl/hadoop-2.7.0

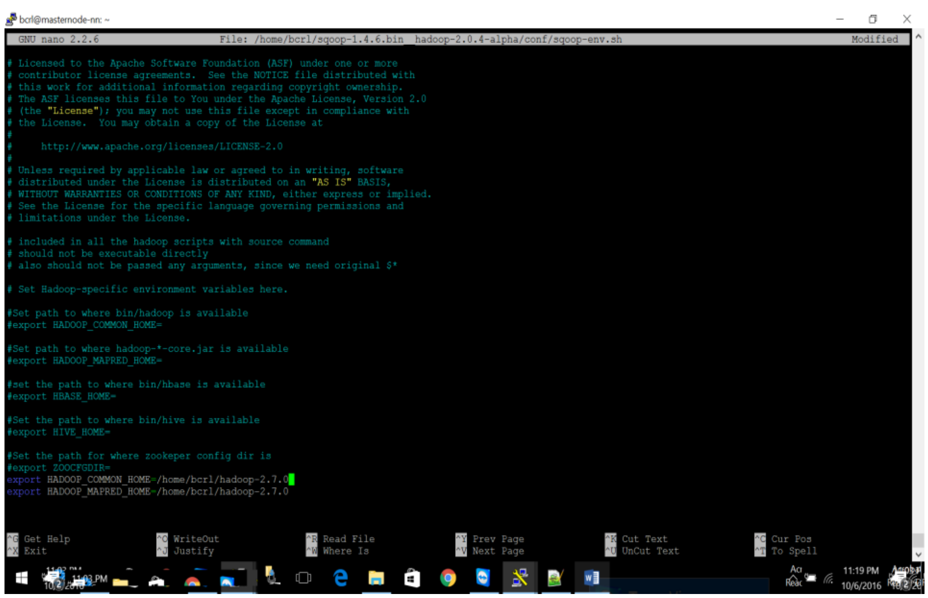


Figure 25. Configuring SQOOP-env.sh

- Configure MySQL-connector-java

Adding MySQL-connector-java.jar to SQOOP libraries.

sudo apt-get install libMySQL-java

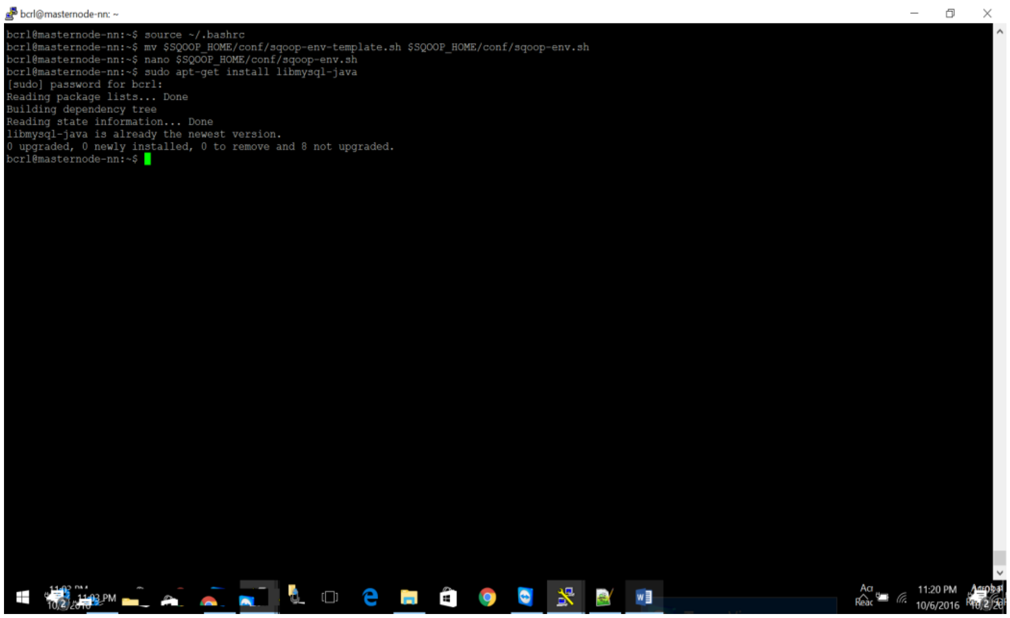


Figure 26. Configuring MySQL–connector-java

ln -s /usr/share/java/MySQL-connector-java.jar $SQOOP_HOME/lib/MySQL-connector-java.jar

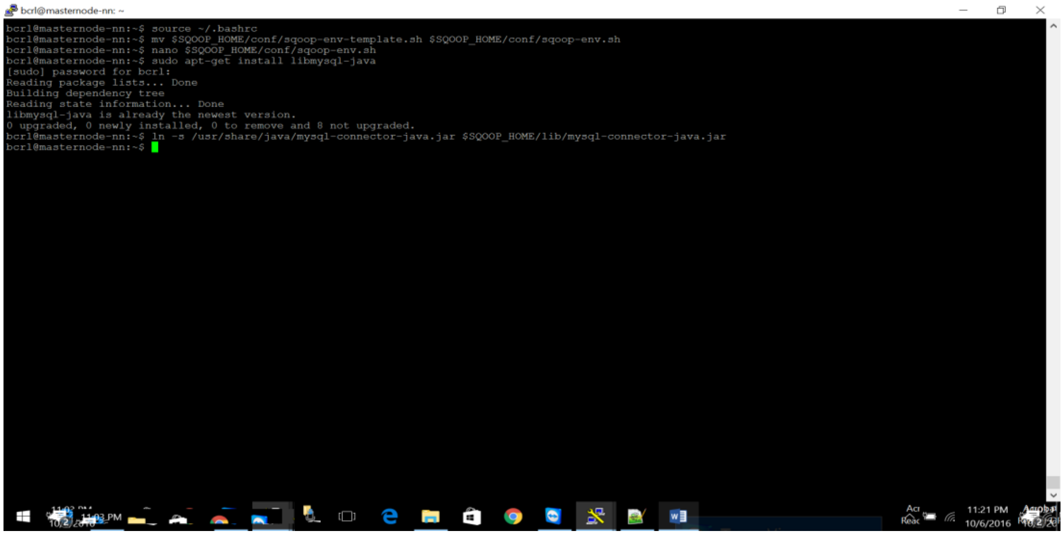


Figure 27. Adding MySQL-connector jar to SQOOP libraries

Verifying SQOOP

The following command is used to verify the SQOOP version.

SQOOP-version

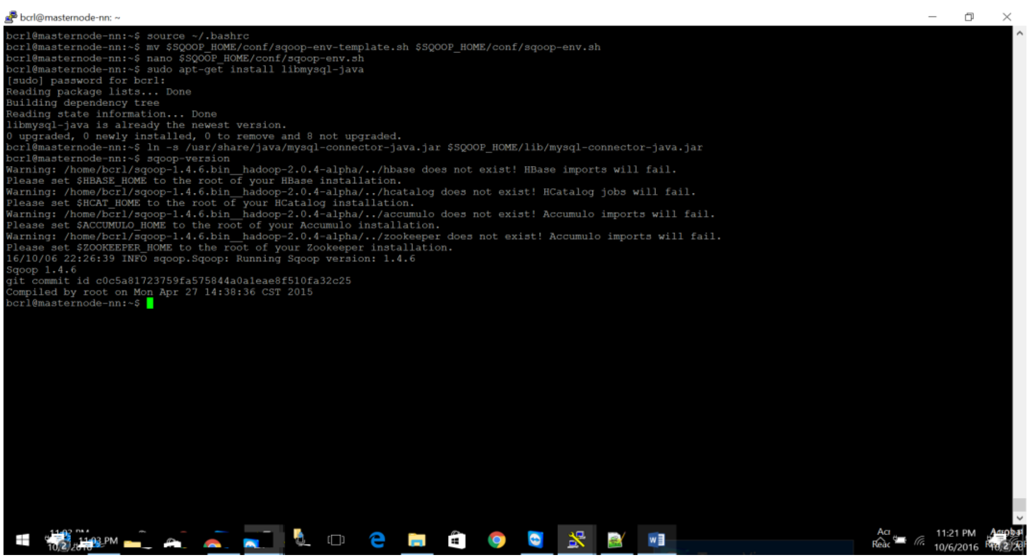


Figure 28. Verifying SQOOP

SQOOP installation is complete.

HIVE Installation:

Apache HIVE is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.

- Downloading HIVE

  Download HIVE binary distribution by following command:

  wget http://ftp.wayne.edu/apache/HIVE/stable/apache-HIVE-1.2.1-bin.tar.gz

  tar xvzf apache-HIVE-1.2.1-bin.tar.gz

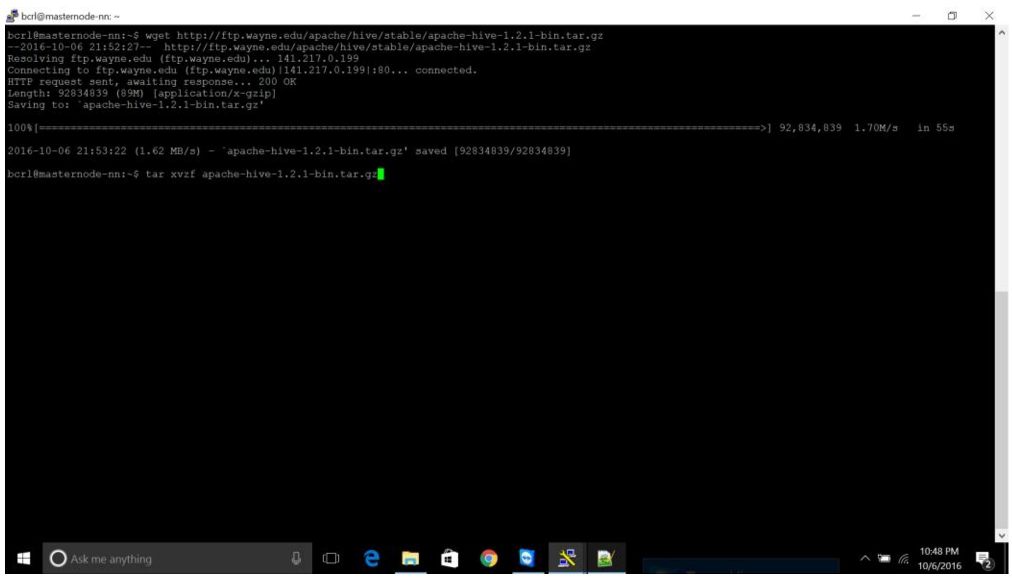


Figure 29. Downloading HIVE

- Installing HIVE

  The following commands are used to extract the HIVE tar ball and move it to "/home/bcrl/HIVE" directory.

  sudo mkdir /home/bcrl/HIVE

  cd apache-HIVE-1.2.1-bin/

sudo mv * /home/bcrl/HIVE/.

- Changing owner and group for HIVE installation directory to Hadoop dedicated user

sudo chown -R bcrl:bcrl /home/bcrl/HIVE

- Configuring bashrc

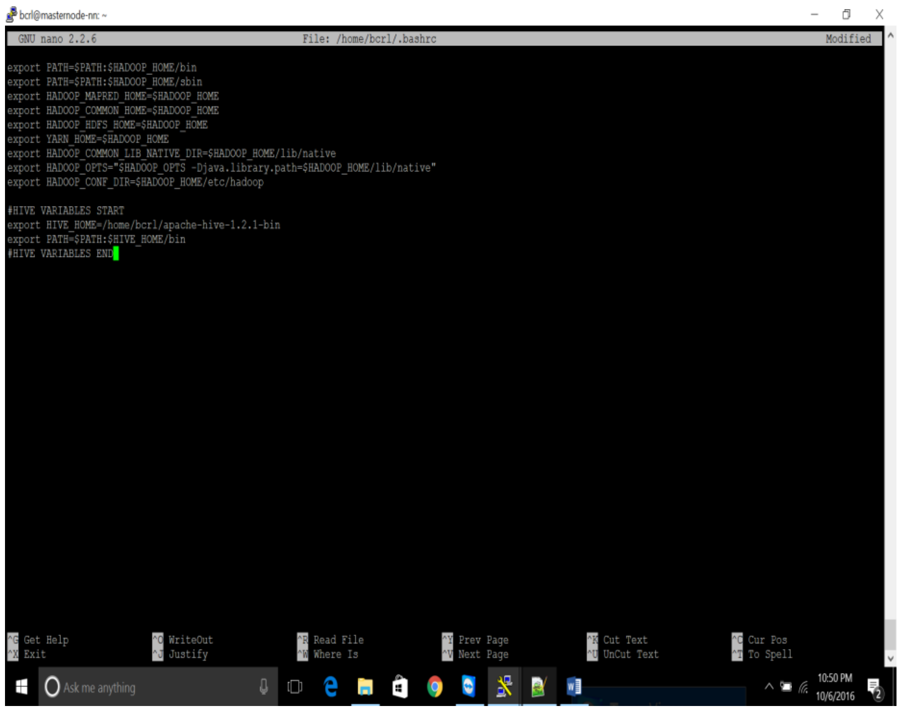


Figure 30. Configuring bashrc in HIVE

nano ~/.bashrc

In bashrc file append the following statements:

#HIVE VARIABLES START

export HIVE_HOME=/home/bcrl/HIVE

export PATH=$PATH: $HIVE_HOME/bin

#HIVE VARIABLES END

source ~/. bashrc

- Configuring HIVE

Open HIVE-config.sh and configure Hadoop home directory path.

nano /home/bcrl/HIVE/bin/HIVE-config.sh

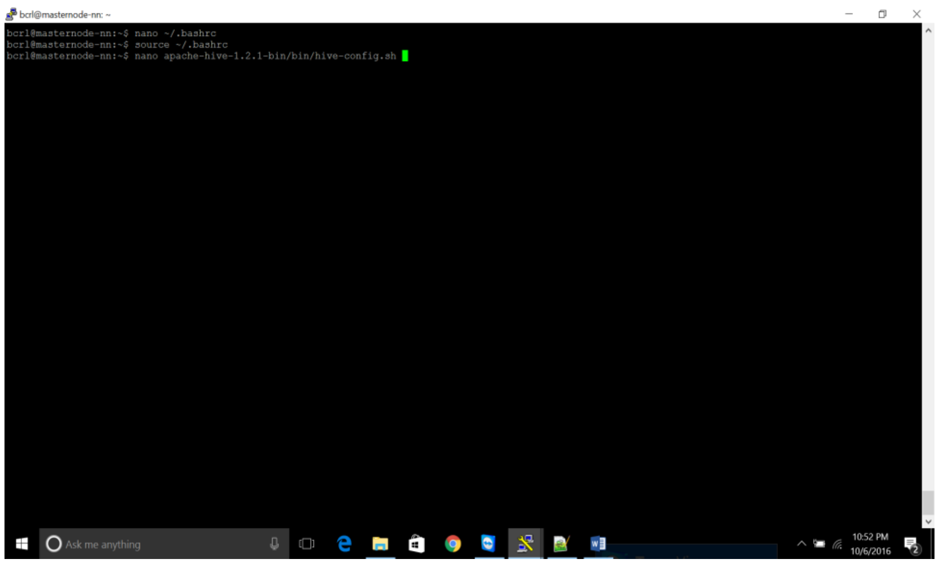


Figure 31. Configuring HIVE

export HADOOP_HOME=/home/bcrl/hadoop-2.0.7

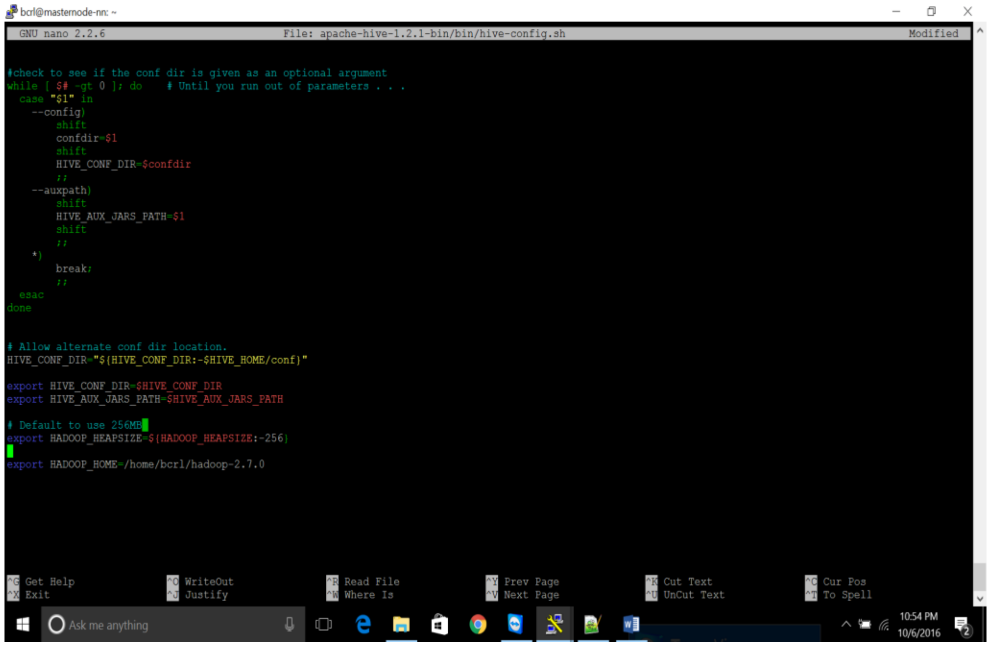


Figure 32. Configuration Commands in HIVE-config.sh

- Configure MySQL-connector-java

Adding MySQL-connector-java.jar to HIVE libraries.

sudo apt-get install lib MySQL-java

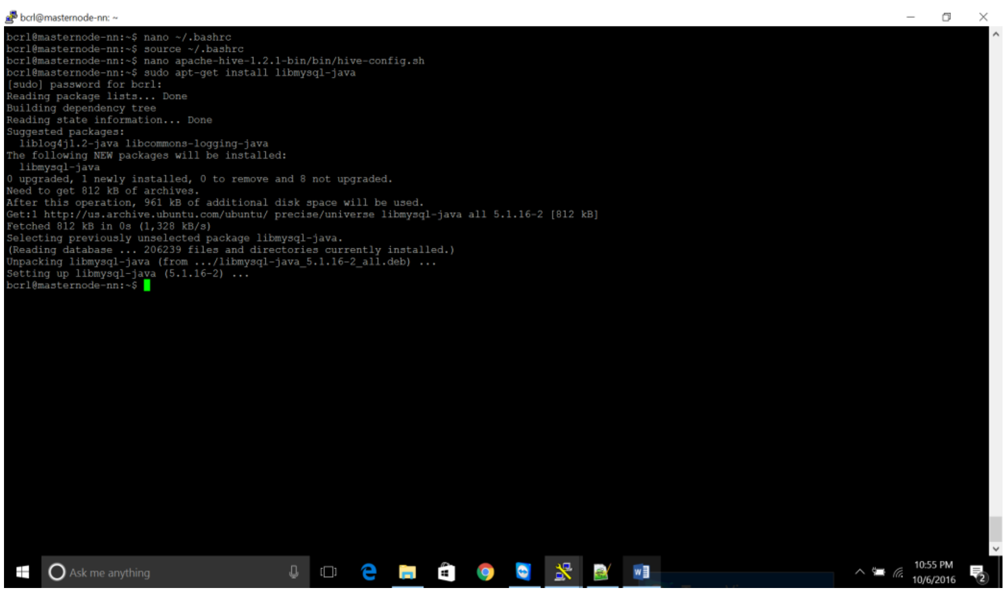


Figure 33. Configure MySQL-connector-java

ln –s /usr/share/java/MySQL-connector-java.jar $HIVE_HOME/lib/MySQL-connector-java.jar

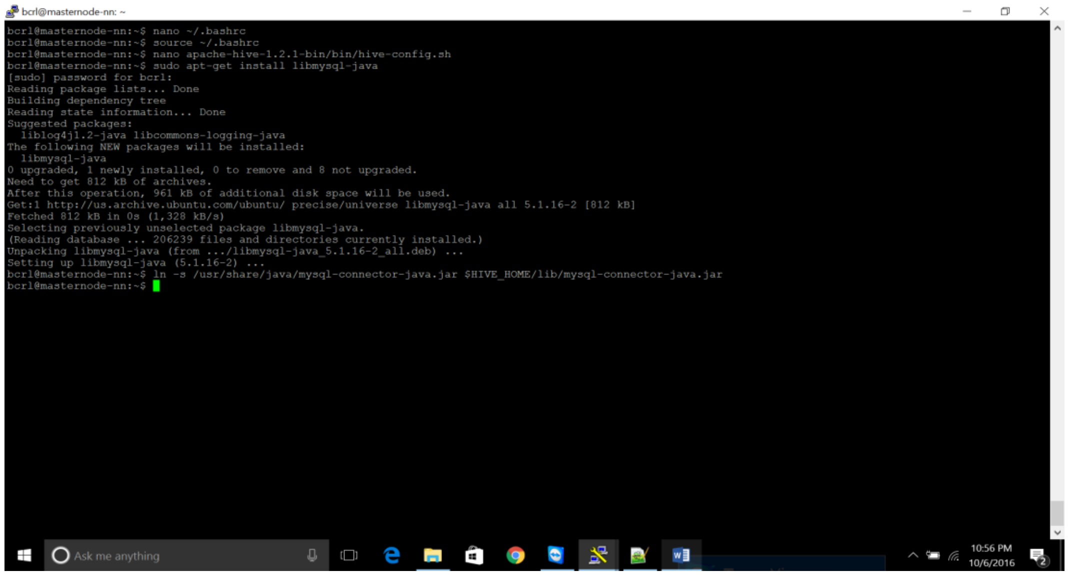


Figure 34. Adding MySQL-connector-java.jar to HIVE libraries

- Configuring Metastore of HIVE

Configuring Metastore means specifying to HIVE where the database is stored. You can do this by editing the HIVE-site.xml file, which is in the $HIVE_HOME/conf directory. First of all, copy the template file using the following command:

sudo mkdir /home/bcrl/HIVE/iotmp

sudo mkdir /home/bcrl/HIVE/iotmp/HIVEjobs

cp /home/bcrl/HIVE/conf/HIVE-default.xml.template

/home/bcrl/HIVE/conf/HIVE-site.xml

Edit HIVE-site.xml and append the following lines between the <configuration> and </configuration> tags:

nano /usr/local/HIVE/conf/HIVE-site.xml

<property>

<name>HIVE.exec. local.scratchdir</name>

<value>/home/bcrl/HIVE/iotmp/HIVEjobs</value>

<description>Local scratch space for HIVE jobs</description>

</property>

<property>

<name>HIVE.downloaded.resources.dir</name>

<value>/home/bcrl/HIVE/iotmp/${HIVE.session.id} _resources</value>

<description>Temporary local directory for added resources in the remote file system. </description>

</property>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:
MySQL://10.31.10.102/metastore_db?createDatabaseIfNotExist=true</value>

<description>metadata is stored in a MySQLserver</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.MySQL.jdbc.Driver</value>

<description>MySQLJDBC driver class</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>HIVEuser</value>

<description>user name for connecting to MySQLserver</description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>HIVEpassword</value>

<description>password for connecting to MySQLserver</description>

</property>

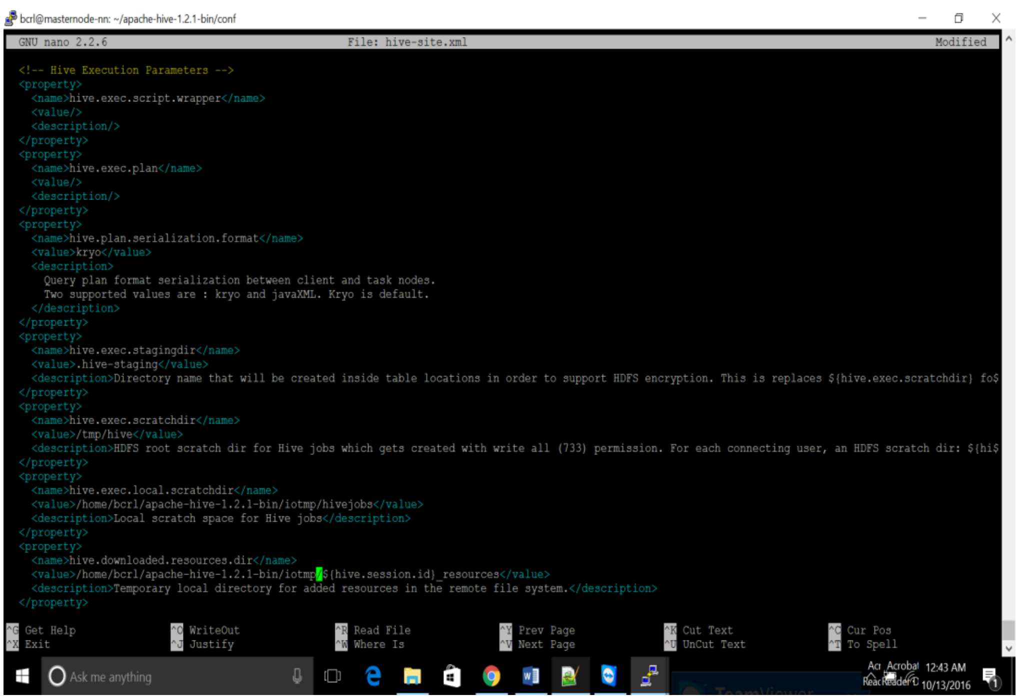


Figure 35. Configuring Metastore of HIVE

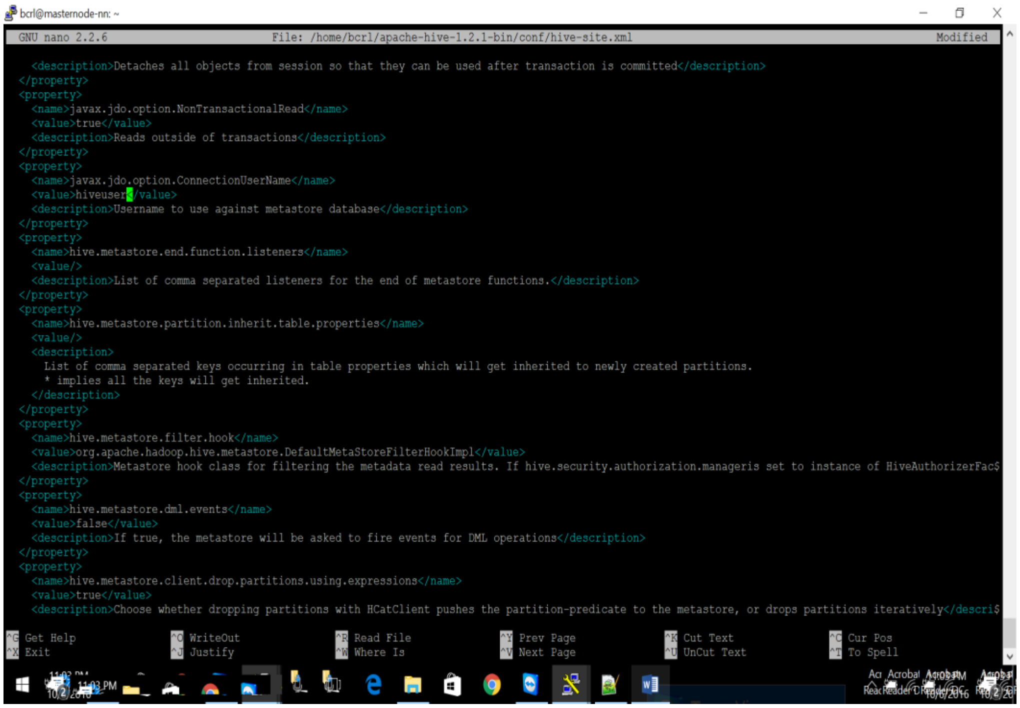


Figure 36. Appending commands for HIVE-site.xml

Configure metastore_db in MySQL

MySQL-u root -p

Enter password: root

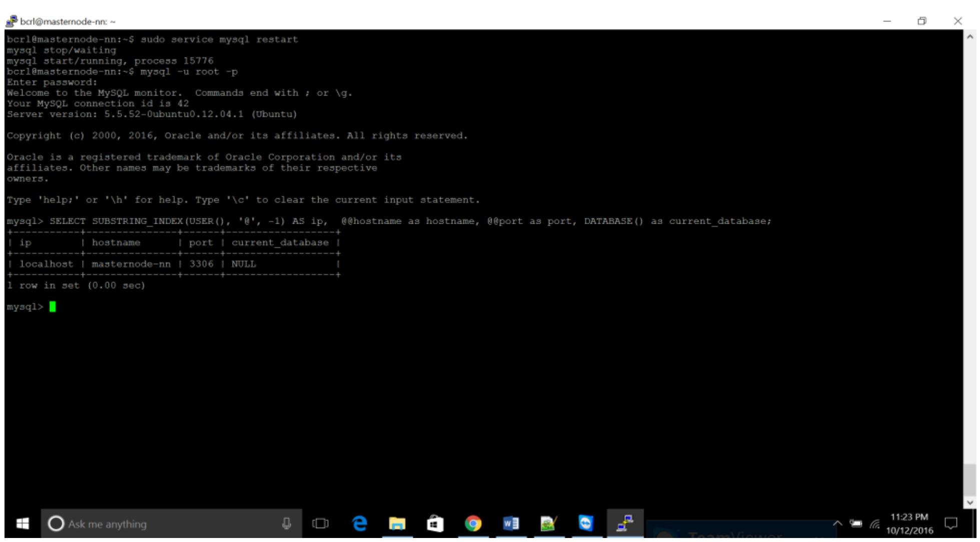


Figure 37. Restarting MySQL

MySQL> create database metastore_db;

MySQL> use metastore_db;

MySQL> SOURCE /usr/local/HIVE/scripts/metastore/upgrade/MySQL/HIVE-schema-0.14.0.MySQL.SQL;

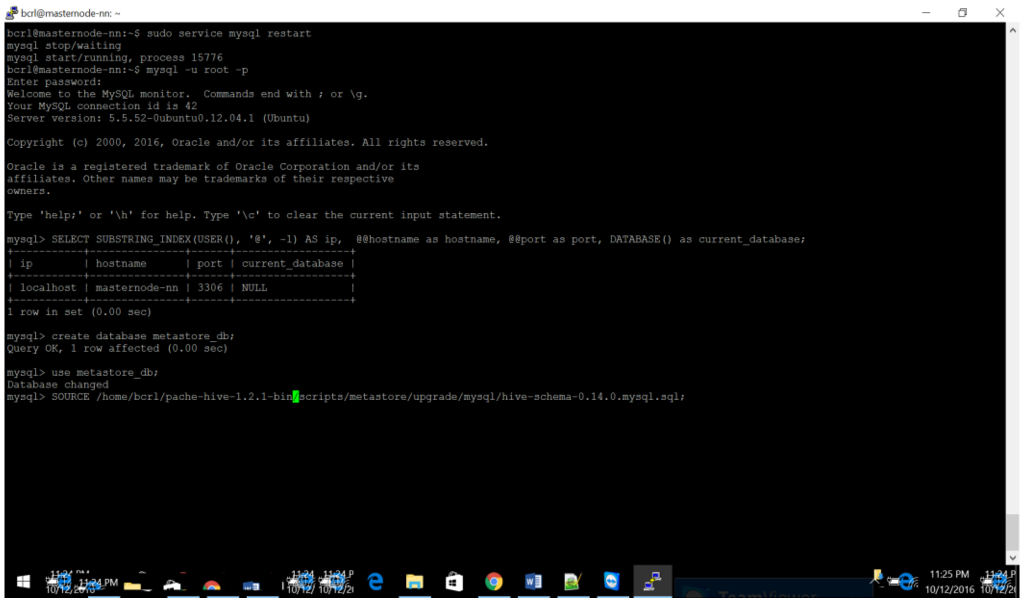


Figure 38. Configure metastore_db in MySQL

MySQL> CREATE USER 'HIVEuser'@'%' IDENTIFIED BY 'HIVEpassword';

MySQL> GRANT all on *. * to 'HIVEuser'@10.31.10.102 identified by 'HIVEpassword';

MySQL> flush privileges;

MySQL> exit;

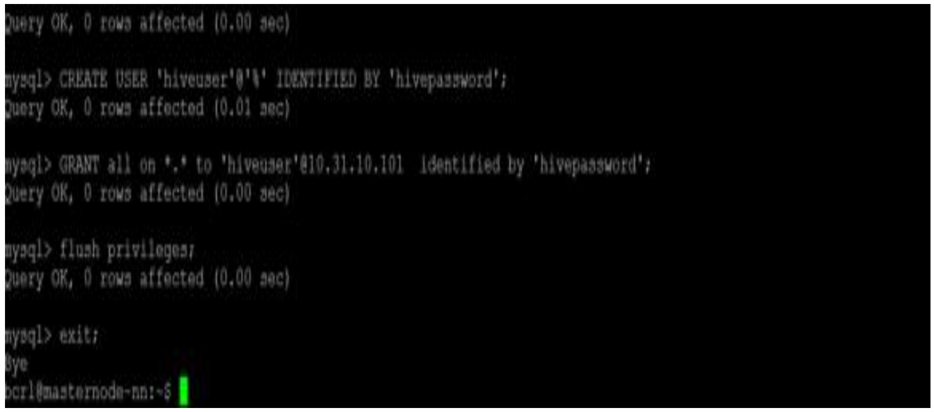


Figure 39. Granting privileges to user

The following command is used to verify the HIVE installation

HIVE

Figure 40. Verifying HIVE installation

Configuring hostname and mapping ip addresses to hostnames

sudo nano /etc/hostname

masternode

sudo nano /etc/hosts

10.31.10.102 masternode localhost

10.31.10.103 DataNode1

10.31.10.104 DataNode2

To check virtual machine hostname

Loading Twitter data from .text file to MySQL

bcrl@masternode: ~$ MySQLimport --user=root --password=root --fields-terminated-by='|' --lines-terminated-by='\n' --local hadoopanalysis TweetData

Importing data from MySQL to HDFS

bcrl@masternode: ~$ SQOOP import --connect jdbc:MySQL://10.31.10.102:3306/hadoopanalysis --table TwitterAnalysis --username HIVEuser --password HIVEpassword

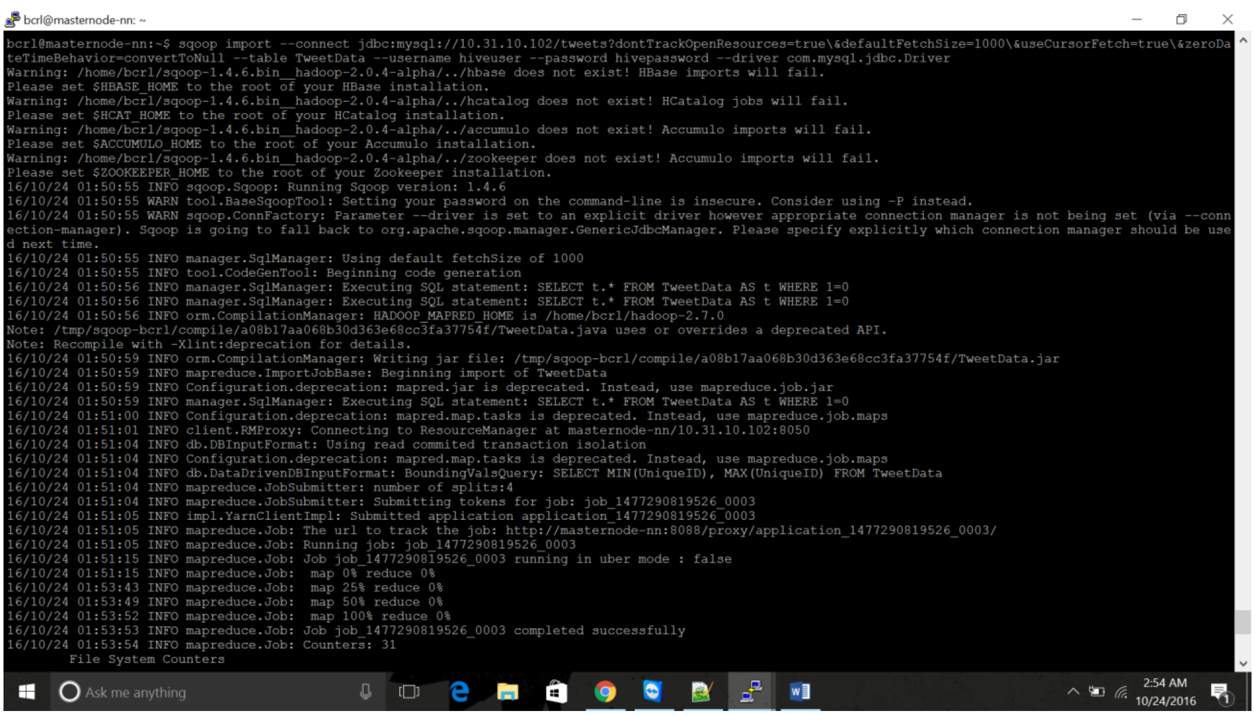


Figure 41. Importing Data from MySQL to HDFS

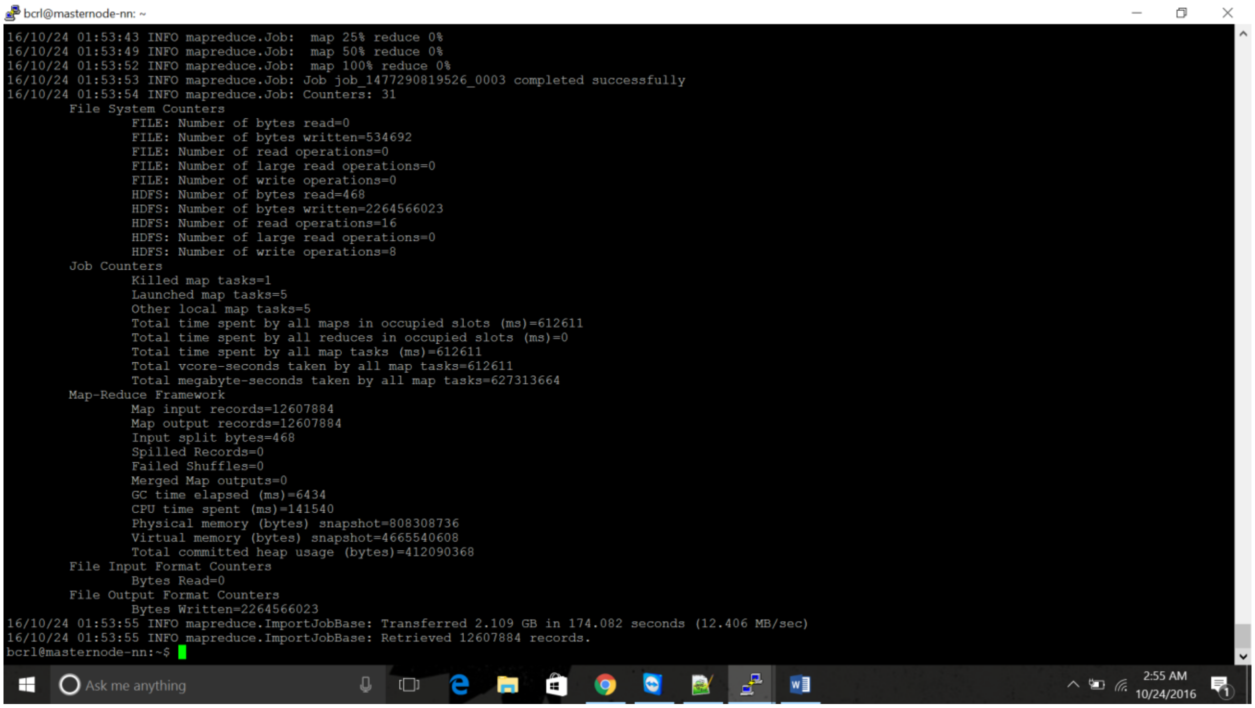


Figure 42. Successfully Installed Hadoop and HIVE

The following command is used to verify the HIVE installation

HIVE>

bcrl@masternode: ~$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa

bcrl @masternode: ~$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys

bcrl @masternode: ~$ ssh-copy-id -i ~/.ssh/id_dsa.pub bcrl@namemode

bcrl @masternode: ~$ ssh-copy-id -i ~/.ssh/id_dsa.pub bcrl@DataNode1

bcrl @masternode: ~$ ssh-copy-id -i ~/.ssh/id_dsa.pub bcrl@DataNode2

Installing and Configuring Apache Hadoop

Hadoop is downloaded from the open source Hadoop source repository by using the following command:

Wget http://www-us.apache.org/dist/hadoop/commom/hadoop-2.6.1/hadoop-2.6.1.tar.gz

After installing the taz.gz file, extract it using

tar xvzf hadoop-2.6.1.tar.gz

Now since Hadoop is installed and extracted, the user needs to be assigned which will be an owner and also change the group for Hadoop installation directory using the following comman

Sudo chown –R bcrl:bcrl /home/bcrl/SQOOP

Finally, Hadoop needs to be configured. There are lots of files that need to be configured in order to configure Hadoop. Some of the configurations are:

1. ~/. bashrc

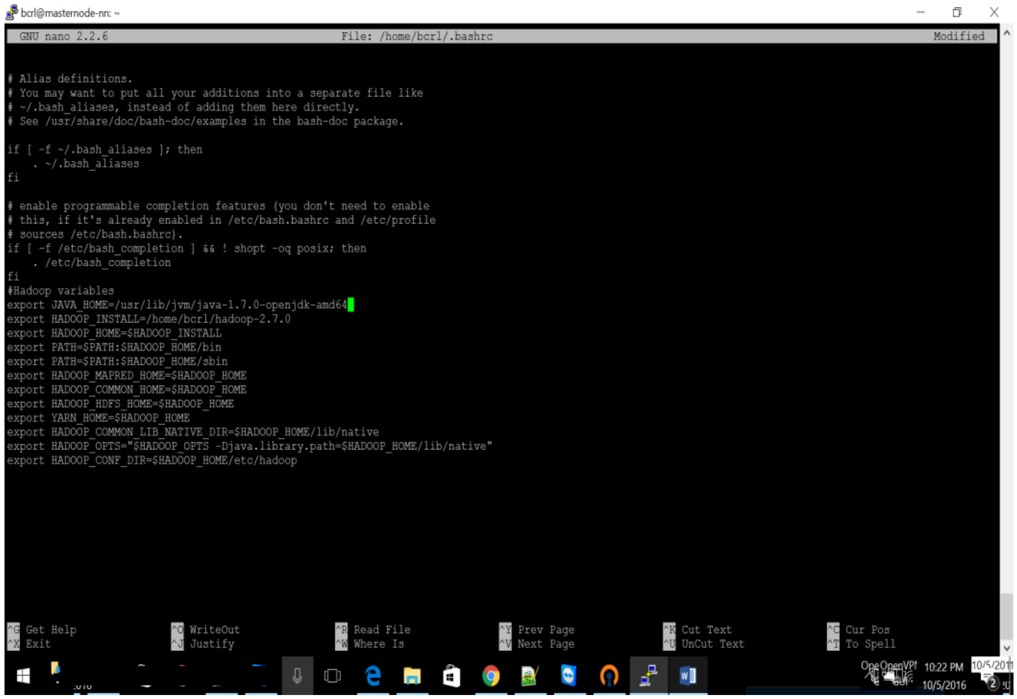


Figure 43. Appending commands in bashrc for Hadoop configuration

#Hadoop variables start

export JAVA_HOME=/usr/lib/jvm/java-6.1-openjdk-amd64

export HADOOP_INSTALL=<Hadoop home directory>

export HADOOP_HOME=$HADOOP_INSTALL

export PATH=$PATH: $HADOOP_HOME/bin

export PATH=$PATH: $HADOOP_HOME/sbin

export HADOOP_MAPRED_HOME=$HADOOP_HOME

export HADOOP_COMMON_HOME=$HADOOP_HOME

export HADOOP_HDFS_HOME=$HADOOP_HOME

export YARN_HOME=$HADOOP_HOME

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native

```
exportHADOOP_OPTS="$HADOOP_OPTS -
Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
#Hadoop variables end
```

Some of the important configurations that need to be done to work with Hadoop are described below with the configuration name and their properties.

1. HDFS-site.xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.NameNode.name.dir</name>
        <value>file:/bcrl/bcrl/hadoop-2.6.1/hadoop_store/HDFS/NameNode</value>
    </property>
 <property>
        <name>dfs.NameNode.http-address</name>
        <value>masternode:51070</value>
   </property>
</configuration>
```

2. yarn-site.xml

```
<configuration>
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>MapReduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services. MapReduce.shuffle.class</name>
        <value> org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>masternode:8026</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>masternode:8031</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>masternode:8051</value>
    </property>
</configuration>
```

3. mapred-site.xml

<configuration>

<property>

<name>MapReduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapred.local.dir</name>

<value>file:/bcrl/bcrl/hadoop-2.6.1/hadoop_store/mapred/local</value>

<description>Determines where temporary MapReduce data is written. It also may be a list of directories. </description>

</property>

<property>

<name>mapred.map.tasks</name>

<value>20</value>

<description>As a rule of thumb, use 10x the number of slaves (i.e., number of TaskTrackers).</description>

</property>

<property>

<name>mapred.reduce.tasks</name>

<value>4</value>

```
<description>As a rule of thumb, use 2x the number of slave processors (i.e., number of TaskTrackers).</description>
</property>
</configuration>
```

4. core-site.xml

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/bcrl/bcrl/hadoop-2.6.1/tmp</value>
<description>A base for other temporary directories. </description>
</property>
<property>
<name>fs. default.name</name>
<value>HDFS://masternode:54310</value>
<description>The name of the default file system.  A URI whose
scheme and authority determine the FileSystem implementation.  The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class.  The uri's authority is used to
determine the host, port, etc. for a filesystem. </description>
</property>
</configuration>
```

5. masters

   bcrl@masternode

6. slaves

   bcrl@DataNode1

   bcrl@DataNode2

Format Hadoop NameNode by following command

hadoop NameNode –format

To start Hadoop daemons, this is the following command

start-all.sh

To stop Hadoop daemons, the following command is used

stop-all.sh

Chapter 5: Analysis and Results

Introduction

This section provides the comparison of the results by performing analysis on both MYSQL and HIVE Query language. This gives a clear result of how the huge amounts of data can be stored in Hadoop HDFS and processed using HIVE with the help of MapReduce and generates results in far less time when compared to the traditional MYSQL database.

Results and Analysis

-Creating Tables in HIVE

HIVE> create table TwitterData(UniqueID BIGINT,TweetID BIGINT, Time_stamp VARCHAR(255), Tweet VARCHAR(255),FavouriteCount BIGINT, ReTweetCount BIGINT, lang VARCHAR(255), UserID BIGINT, UserName VARCHAR(255), ScreenName VARCHAR(255),Location VARCHAR(255), FollowersCount BIGINT, FriendsCount BIGINT, Statuses BIGINT, Timezone VARCHAR(255));

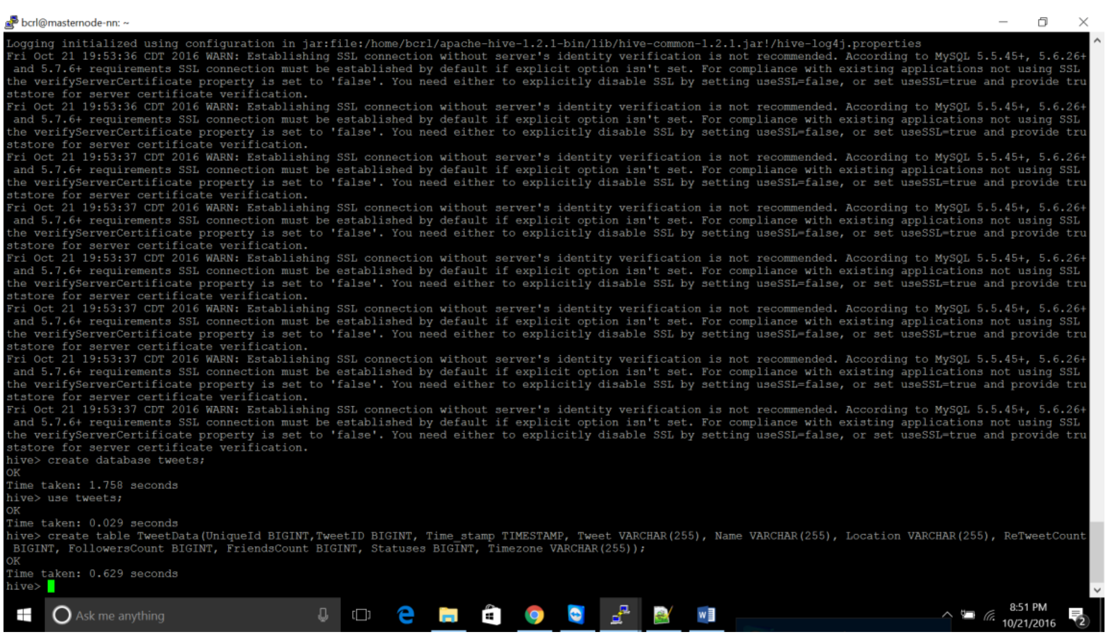


Figure 44. This shows the time taken to perform this query for 0.629 seconds.

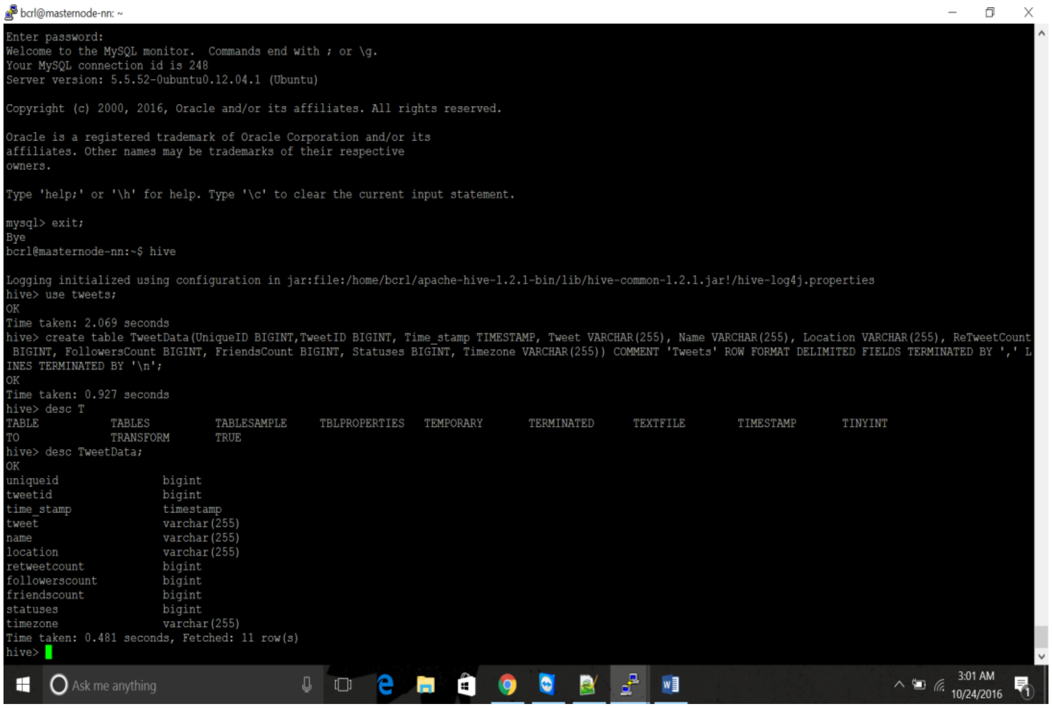


Figure 45. The time taken to perform this Query in HIVE is 0.4 seconds

Table 3

TwitterData table in MySQL and HIVE

Field	Type
UniqueID	Bigint
TweetID	Bigint
CreatedAt	Varchar
Tweet	Varchar
FavouriteCount	Bigint
ReTweetCount	Bigint
Lang	Varchar

UserID	Bigint
UserName	varchar
ScreenName	varchar
Location	varchar
FollowersCount	Bigint
FriendsCount	Bigint
Statuses	Bigint
Timezone	Varchar

Loading data from HDFS to HIVE tables:

HIVE> load data inpath '/user/bcrl/TweetData' into table TweetData;

A table of results for the experimental trials appears below. In both cases SQL like code was used to define the query. In the MySQL database, basic SQL was used in the Hadoop file system HIVE was used as a front-end and therefore, the HIVE version of SQL was utilized.

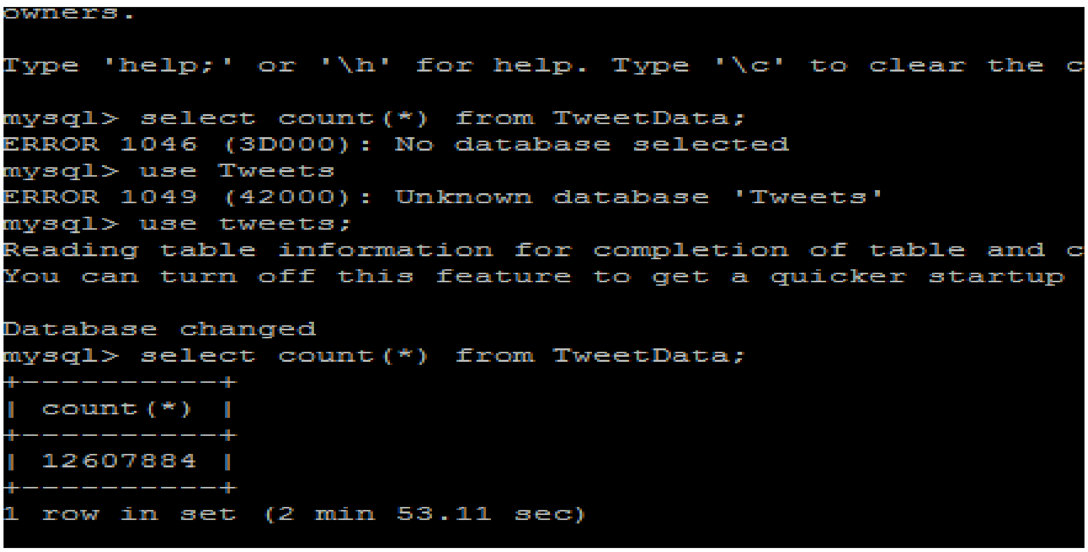
Table 4

Comparison Results of HIVE and MySQL

Query	HIVE Computation Time	MySQLComputation Time
Select * from TweetData;	7min 32sec	11min 53sec

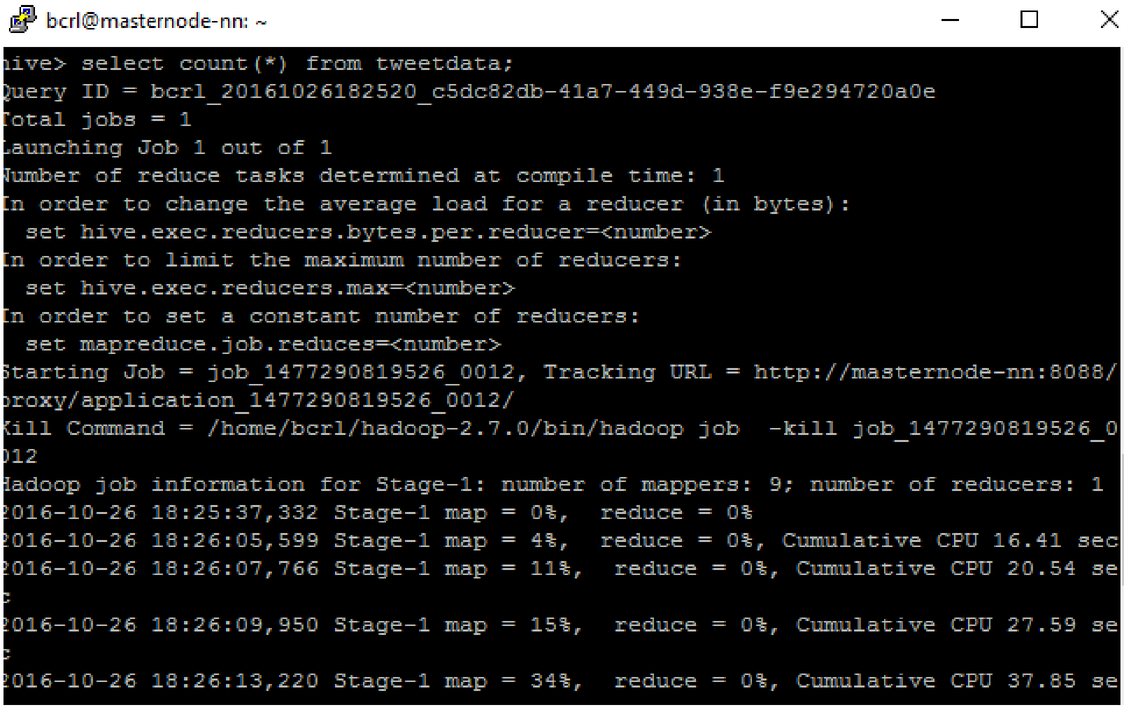
Select count(*) from TweetData;	1min 53sec	2min 35sec
Select count(Distinct UniqueId) from TweetData;	123.279 sec	2 min 32 sec

HIVE QUERIES VS MYSQL



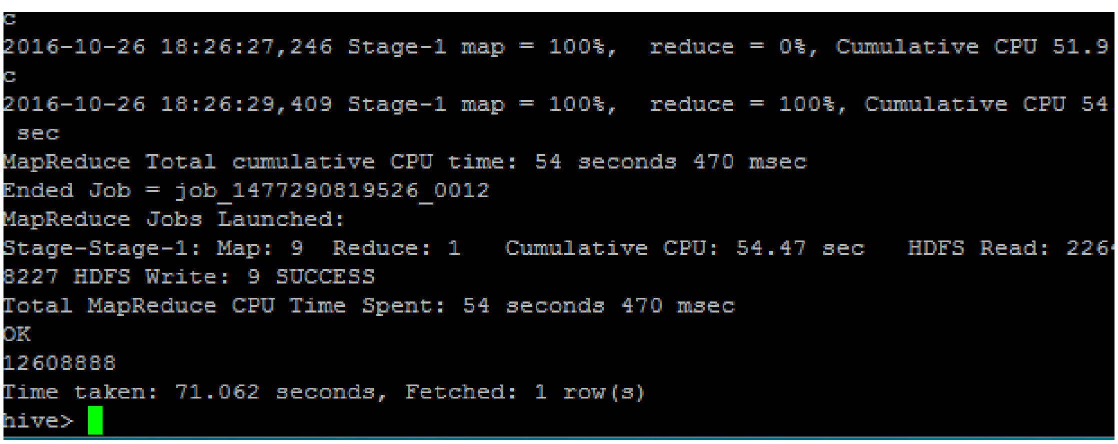
```
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the c

mysql> select count(*) from TweetData;
ERROR 1046 (3D000): No database selected
mysql> use Tweets
ERROR 1049 (42000): Unknown database 'Tweets'
mysql> use tweets;
Reading table information for completion of table and c
You can turn off this feature to get a quicker startup

Database changed
mysql> select count(*) from TweetData;
+----------+
| count(*) |
+----------+
| 12607884 |
+----------+
1 row in set (2 min 53.11 sec)
```

Figure 46. MySQL performance for counting the TweetData is shown which is 2 min 53.11 sec



```
bcrl@masternode-nn: ~
hive> select count(*) from tweetdata;
Query ID = bcrl_20161026182520_c5dc82db-41a7-449d-938e-f9e294720a0e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1477290819526_0012, Tracking URL = http://masternode-nn:8088/
proxy/application_1477290819526_0012/
Kill Command = /home/bcrl/hadoop-2.7.0/bin/hadoop job  -kill job_1477290819526_0
012
Hadoop job information for Stage-1: number of mappers: 9; number of reducers: 1
2016-10-26 18:25:37,332 Stage-1 map = 0%,  reduce = 0%
2016-10-26 18:26:05,599 Stage-1 map = 4%,  reduce = 0%, Cumulative CPU 16.41 sec
2016-10-26 18:26:07,766 Stage-1 map = 11%,  reduce = 0%, Cumulative CPU 20.54 se
c
2016-10-26 18:26:09,950 Stage-1 map = 15%,  reduce = 0%, Cumulative CPU 27.59 se
c
2016-10-26 18:26:13,220 Stage-1 map = 34%,  reduce = 0%, Cumulative CPU 37.85 se
```

Figure 47. The same query (count) number of TweetData is shown here



```
c
2016-10-26 18:26:27,246 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 51.9
c
2016-10-26 18:26:29,409 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 54
 sec
MapReduce Total cumulative CPU time: 54 seconds 470 msec
Ended Job = job_1477290819526_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 9  Reduce: 1   Cumulative CPU: 54.47 sec   HDFS Read: 226
8227 HDFS Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 54 seconds 470 msec
OK
12608888
Time taken: 71.062 seconds, Fetched: 1 row(s)
hive>
```

Figure 48. This shows the time which is 79.062 seconds.

This clearly proves the objective of the paper.

❖ In this paper, the data is first generated from a Twitter API and then loaded into MySQL. Also, the SSH keys are authenticated and configured by using the DSA algorithm to ensure security between virtual machines while transferring data. Second, Apache Hadoop was installed and configured in all three virtual machines. In which two virtual machines acts as DataNodes and one virtual machine acts as the NameNode.

- Then the data is exported into HDFS using the tool SQOOP and HIVE is installed on top of Hadoop and created tables in HIVE data warehouse and then transfers data into HIVE tables.
- Finally, performance is tested in the tables using both the databases (MYSQL, HIVE) and could show which gives the better performance.

Summary

This section explains how the data is used and analyzed. It also presents the implementation of parallel processing of data with the Twitter data set. The next section concludes the paper along with the future work that can be done.

Chapter 6: Conclusion and Future Work

Conclusion

The literature indicates that processing Big Data in a reasonable time frame can be a challenging task. One of the most promising platforms is the concept of Exascale computing. This study created a testbed based on recommendations for Big Data within the Exascale architecture. First, this was easily accomplished within the private cloud using VMware across a cluster of devices. Second, because regular commodity components could be used this was a cost-effective solution. Third, because the HIVE front end was SQL-based and I had a background in SQL the learning curve to take advantage of this system was minimal. Last, HIVE integrates directly to the MapReduce function so implementing the parallel processing within Hadoop was easily accomplished.

The literature also indicated that traditional databases were designed for transactional processing and work well in instances where a single record needs to be read, written or updated. Hence, a database may grow over time, but slowly. So, therefore, it is easier to get data in a traditional database than out. The experimental trials carried out herein confirmed that fact and illustrated the advantages of distributed file system when large amounts of data need to be accessed.

Accessing all the records in TweetData logs illustrated that a distributed file system could be about 30% faster. It would be expected that the underlying hardware used for the Hadoop file system could be expanded and tuned for better performance because the test-bed only included three nodes. The additional resources within a distributed file system not only would allow faster processing but solve problems not possible in the traditional architecture.

As the field of Big Data matures and grows the need to process larger and larger amounts of data in a timely manner will continue to be a concern. The Exascale computing architecture offers a promising and cost-effective platform to address that concern. Distributed file systems such as Hadoop offer a relatively simple means of taking advantage of the parallel processing required within distributed file systems.

Future Work

I found Hadoop easy to configure, use and adapt in solving their Big Data needs. However, further research is needed that uses larger data sets and more complex queries to truly assess the capabilities of distributed file systems. Accordingly, research related to optimizing the number of nodes and the intercommunication paths in the underlying infrastructure will be needed as well.

The problem with a MySQL database is that as the data grows larger, the time required to process the data increases and additional resources may be required. With Hadoop, HIVE, and Pig processing time can be faster than MySQL. The data model in this paper is taken from Twitter showed that HIVE is more appropriate for this data model in a low-cost environment. Now, when big organizations use the same technique for analysis there will be other issues to be considered as well. As the data increases and network traffic increases, network and system administrators can face serious problems around Big Data network traffic. Network traffic data can be stored in structured or unstructured format. However, RDBMS were not designed to store and process unstructured data. HIVE stores the data in tables like relational database management systems. However, there needs to be some traffic querying and analyzing systems that handle TCP and UDP analysis of big network traffic data and reduce the false positive detection rate with accuracy detention rate of the attacks to the network security system. Hence the performance of Big Data technologies on Big Data network traffic system can be done in the future.

Etikala, Sultana, Mark, Beche, and Guster (2015) described the security challenges in Big Data which supports my thoughts. While Big Data appears to be an established field it is still emerging. It can be expected that a means of improving the storage solutions, access times, security and optimizing software will be topics explored by data scientists (Najafabadi et al., 2015). While Big Data provides a wealth of decision-making power because of the massive volume of data it uses its original security design was overly simple. In other words, the data was protected only by the fact that data gathering on that scale was difficult and can now be easily violated (Weber, 2012). In part, security within Big Data is becoming more important due to emerging technologies such as Cloud Computing, analytics engines, and social networks. This environment creates a complex research challenge which necessitates the development of secure big data models. Several techniques and algorithms have been proposed recently, mostly adhering to algorithmic paradigms or model-oriented paradigms (Cuzzocrea, 2014).

References

Apache Hadoop Wiki, n.d. Hadoop.Apache.org

Apache SQOOP, http://sqoop.apache.org/

Borthakur, D. (2007). The Hadoop Distributes File System: Architecture and Design taken from https://svn.apache.org/repos/asf/hadoop/common/tags/release-0.16.4/docs/hdfs_design.pdf

Cecchinel, C., Jimenez, M., Mosser, S., & Riveill, M. (2014). An Architecture to Support the Collection of Big Data in the Internet of Things. IEEE World Congress on Sercices, pp.442-449.

Cuzzocrea, A. (2014). Privacy and Security of Big Data: Current Challenges and Future Research Perspectives. Proceedings of the First International Workshop on Privacy and Security of Big Data, ACM, pp. 45-47.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach., D. A., Burrows, M., Chandra, T., . . . Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems 26, 2, 4:1-4:26.*

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *"In: OSDI '04: 6TH Symposium on Operating Systems Design and Implementation (USENIX and ACM SIGOPS), pp.137-150.*

Dean, J., & Ghemawat. S. (2010). MapReduce: A flexible Data Processing Tool.Communications of the ACM, *Vol.53 No.1, Pages 72-77.*

Etikala, P., Sultana, A., Schmidt, M., Beche, G. D., & Guster, D. (2015). Using Hadoop to Support Big Data Analysis: Security Concerns and Ramifications.

Etikala, P. (2016). Designing & implementing a java web application to interact with data stored in a distributed file system, Department of Information Assurance, SCSU

Fang, W., Sheng, V. S., Wen, X., & Pan, W. (2014). Meteorological Data Analysis Using mapreduce. The Scientific World Journal, Volume 2014 (2014), Article ID 646497, 10 pages, http://dx.doi.org/10.1155/2014/646497

Fan, J., Han, F., & Liu, H. (2014), Challenges of Big Data Analysis Natl Sci Rev (2014) 1 (2): 293-314

Fuad, A., Erwin, A., & lpung, H. P. (2014). Processing Performance on Apache Pig, HIVE and MySQL from http://ieeexplore.ieee.org/document/7010600/

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2014), The rise of "big data" on cloud computing: Review and open research issues Volume 47, January 2015, Pages 98–115

IBM. (2015A). The four v's of Big Data. Retrieved March 11, 2016 from https://www-01.ibm.com/software/data/infosphere/hadoop/HIVE/

IBM. (2015B). Why speed matters for big data and analytics. Retrieved Feburary 12, 2016 from http://www-01.ibm.com/software/data/infosphere/hadoop/pig/.

Jewell, D., Barros, R. D., Diederichs, S., Duijvestijn, L. M., Hammersley, M., HazrA, A., . . . & Zolotow, C. (2014). Performance and Capacity Implications for Big Data. Redpaper, ibm.com/redbooks.

Jacobs., A. (2009). The Pathologies of Big Data. Queue – Data. Vol. 7 Issue 6

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014).

Big Data and Its TechnicalChallenges. Communications of the ACM, Vol. 57 No. 7, Pages 86-94.

Jarr, Scott (2014). Part Three: Designing a Data Architecture to Support Both Fast and Big Data. https://voltdb.com/blog/part-three-designing-data-architecture-support-both-fast-and-big-data-0

Kumar, R., Gupta, N., Charu, S., Bansal, S., & Yadav, K. (2014). Comparison of SQL &HIVEQL International Journal for Research in Technological Studies| Vol. 1, Issue 9, August 2014.

Levy, E., & Silberschatz, A. (1990), Distributed File Systems: Concepts and Examples. ACM Computing Surveys, Vol. 22, No. 4, December 1990

Lucas, R., Ang, J., Bergman, K., Borkar, S., Karlson, W., Carrington, L., . . . & Stevens, R. (2014). *Top Exascale Research Challenges. Office of Science, U.S Department of Energy, Washingtion, D.C.*

Michael, O. (2011). When Slower is actually faster. https://blog.macsales.com/11825-when-slower-is-actually-faster

MySQL Wiki, https://en.wikipedia.org/wiki/MySQL

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. N., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics, Journal of Big Data, 2:1.

Pol, U. R. (2016). Big Data Analysis: Comparison of Hadoop, Map Reduce, Pig and HIVE. International Journal of Innovative Research in Science, Engineering and Technology (An ISO 3297: 2007 Certified Organization) Vol. 5, Issue 6, June 2016.

Quintero, D., Navarro, E. A., Garro, P. B., Castro, R. C. F. D., Huertas, L. C. C., Jiang, P., . . .&, J. (2015). Implementing an IBM InfoSphere BigInsights Cluster Using Linux or Power. In IBM Redbooks,Pg. 28.

Rao, B. T., Sridevi, N. V., Reddy, V. K., & Reddy, L. S. S. (2012). Performance Issues of Heterogeneous Hadoop clusters in Cloud Computing.

Global Journal of Computer Science and Technology, Volume XI Issue VIII May 2011

Reed, D. A., & Dongarra, J. (2015).

Exascale Computing and Big Data, Communications of the ACM, Vol. 58 No. 7, Pages 56-68.

Sqoop User Guide, (v1.4.5)

https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html

Schmid, P. (2006) Capacity Outrage performance taken from

http://www.tomshardware.com/reviews/15-years-of-hard-drive-history,1368-2.html

Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., & Rasin, A. (2010). MapReduce and Parallel DBMSs: Friends or Foes? Communications of the ACM, Vol. 53 No. 1, Pages 64-71.

Sultana, A., & Etikala, P. (2015). Independent Study for class IA 659, Big Data Analysis.

Shvachko, K., Kuang, H., & Radia, S. (2010). The Hadoop Distributing File System from

http://ieeexplore.ieee.org/abstract/document/5496972/authors

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., . . .& Murthy, R. (2009). HIVE: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, 2(2), 1626-1629.

White, T. (2012). Hadoop: The Definitive Guide (*Third Edition*), Sebastopol CA.

Weber, S. (2012). Big Data Privacy and Security Challenges. Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security, ACM pp. 1-2.