PERFORMANCE EVALUATION OF BIG DATA PLACEMENT STRUCTURES IN MAPREDUCE-BASED DATA WAREHOUSE SYSTEMS

A Thesis

Presented to

The Faculty of the College of Graduate Studies

Lamar University

In Partial Fulfillment

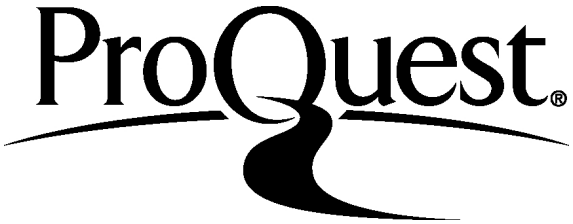
of the Requirements for the Degree

Master of Science in Computer Science

by

Mohammad Rakibul Hasan

May 2016



PERFORMANCE EVALUATION OF BIG DATA PLACEMENT STRUCTURES IN MAPREDUCE-BASED DATA WAREHOUSE SYSTEMS

MOHAMMAD RAKIBUL HASAN

Approved:

_____
Kami Makki
Supervising Professor

_____
Stefan Andrei
Committee Member

_____
Jiangjiang (Jane) Liu
Committee Member

_____
Stefan Andrei
Chair, Department of Computer Science

_____
Joe Nordgren
Interim Dean, College of Arts and Science

_____
William E. Harn
Dean, College of Graduate Studies

ABSTRACT

PERFORMANCE EVALUATION OF BIG DATA PLACEMENT STRUCTURES IN MAPREDUCE-BASED DATA WAREHOUSE SYSTEMS

by

Mohammad Rakibul Hasan

The size of data sets is growing rapidly, which requires fundamentally innovative techniques and technology to capture, store, distribute, and process promptly and cost effectively. Hadoop software framework with high-performance execution engines (MapReduce) is capable of processing large data sets across clusters that provide scalable and fault-tolerant capability on distributed systems. MapReduce-based warehouse system with data storage format is very useful for data summarization and query analysis. The warehouse system can contain millions of row column value and therefore, data placement structure plays a significant role that can influence the warehouse performance. In this research, we examined the performances of Hive's data file formats, the RCFile and ORCFile on top of Hadoop. For this experiment, we design and implement a distributed cluster by three nodes master-slave architecture, where we store and organize the data according to the above files' format structure. We investigate the file format efficiency in terms of data loading, data storage and query processing using MapReduce. The experimental results can lead to choosing the perfect and useful file format for a data warehouse system for Big Data processing

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my thesis advisor Dr. S. Kami Makki for coming up with this topic and steering me towards my final objective. I also want to show my appreciation to Dr. Stefan Andrei and Dr. Jiangjiang (Jane) Liu, the other members of the committee. I thank my parents, my wife Moushumy Khatun, my undergraduate thesis supervisor Mohammad Shamsul Arefin, and my friend P.M Mahmudul Hasan for constantly supporting and motivating me with their best wishes. Finally, I would like to thank the Almighty Allah for helping me to believe in myself and giving me the courage to complete my work.

Table of Contents

List of Tables

Table Page

List of Figures

Chapter 1

Introduction

1.1 Big Data Overview

We are entering in technological advancement era, where every sector is now generating an unprecedented amount of data. Digitalization of every device and escalating the number of Internet users help to grow the data exponentially that makes the traditional data processing technology obsolete. According to ACI in 2012, 2.5 Exabyte's of data were generated in every day (Gunelius 2014), and the total volume of the data in the world is doubled every two years (NTTDATA 2015). Statistics show that two billion people connected to the Internet in 2015, which is 100 times more than 1995 ("Internet Live User" 2014).

The Web 2.0 is another important factor where users can collaborate and share their experiences in online such as blogs, wikis, social networking, web application, video sharing and all types of web services(Wikipedia 2016b). Also, different sources such as social media are playing a paramount role in generating these massive data. For example in every minute Google receives 4 million queries, Facebook users share 2.5 million content, Twitter users tweet nearly 300,000 times, Instagram users post nearly 220,000 new photos, YouTube users upload 72 hours of new video content, Apple users download nearly 50,000 apps, email users send over 200 million messages and so on (Gunelius 2014). The stunning growth rate of this enormous amount of data makes the data big. Simply, Big Data is a large data set with full of complexity. Big Data has defined by four V's, which are volume, variety, velocity and value (Oracle Corporation 2015).The following characteristics measure the data as a Big Data.

- Volume: Volume indicates the amount of data which is generated in the system, where this data has unknown value and low density. The data volume is growing very fast, for example, can be tens of terabytes to hundreds of Petabytes.
- Velocity: Velocity refers the speed rate at which the data are generating and evaluating in real time manner to meet the demand and challenges.
- Variety: The massive amount of data is gathering with huge speed, and this data can be of different types and nature such as structured, semi-structured and unstructured data. The data sources are also sending changes in real time without notice, which creates a huge burden for processing and analyzing.
- Value: There are a great insight and intrinsic value in every piece of data. However, the challenge is to find out the real patterns, making a perfect assumption with predicting behavior from the huge volume of data, which may lead the confident decision making for cost reduction and reduced risk.

Besides this definition of Big Data by four V's there are also some extra characteristics (SAS 2015) that we can define which are following:

- Veracity: This indicates the quality of capture data, which totally depend on the veracity of the source data.
- Complexity: When a vast amount of data with a great speed are coming from multiple sources then it is a real challenge to manage, and it should be linked and connected to the user so that end users can find their relevant information. Figure 1 shows the definition and nature of Big Data (Corrigan 2012).

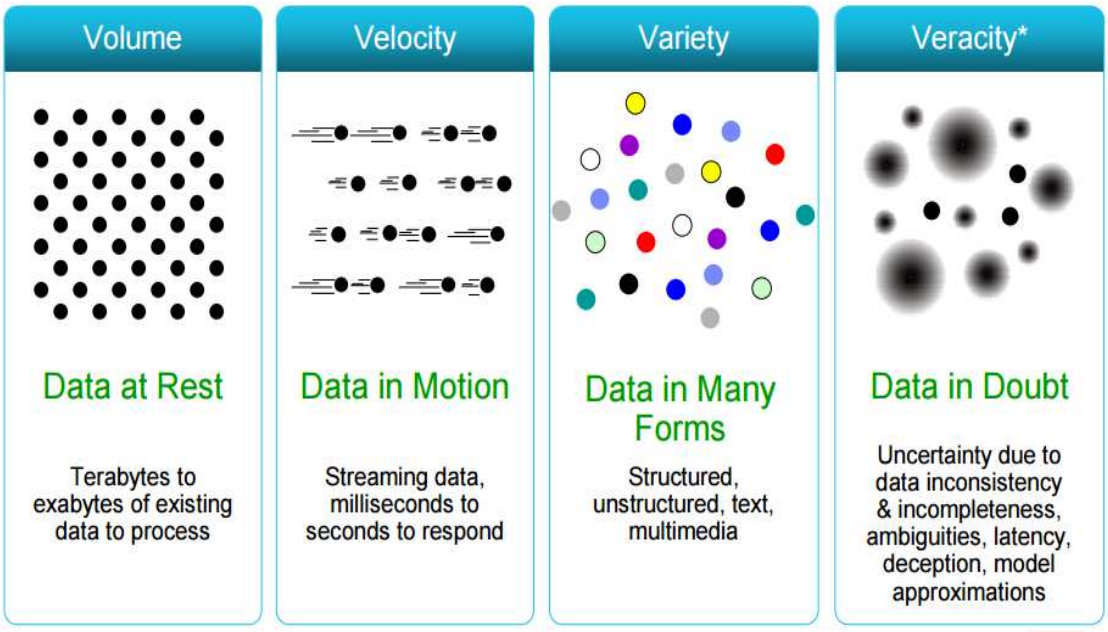


Figure 1. Big Data definition and characteristics.

Big Data solution requires new architecture, techniques, and algorithms to avoid excessive operations of the traditional database system. To store, process and integrate Big Data every company experiences different types of technology, because Big Data reveals the truth, helps to analysis the plan, increase brand value and product reliability (Oracle Corporation 2015). There are two leading Big Data technology namely Apache Hadoop and NoSql database.

Apache Hadoop and its open source framework are one of the best solutions of cluster computing for providing scalable and fault tolerant structure of Big Data analysis. Hive and Pig are two most popular and effective data warehouse system for Facebook and Yahoo respectively (He et al. 2011). However, these warehouse systems do not

directly control the storage in the cluster system. The files system of Hadoop for storage of huge amount of data called Hadoop Distributed File System (HDFS).

Based on analysis of Facebook Data Infrastructure team, in Hadoop MapReduce system there are four critical factors that are affecting the data placement structure of Big Data (He et al. 2011).

1. Fast data loading: Data is generating from everywhere from different users and different systems. In everyday Facebook, accepts more than 20 Terabytes of data. Therefore, it is highly required that Facebook data warehouse stores the data, which effectively reduces the data loading time.
2. Fast Query Processing: Queries can be either in real time web request or offline, where users submit heavy decision-making queries. Therefore, data placement structure needs to process these queries with a very high speed to meet the demand of users' satisfaction.
3. Highly efficient storage space utilization: To meet the demand for growing users' activities Big Data requires a highly efficient scalable storage capacity and computing power.
4. Strong adaptability to highly dynamic workload pattern: Different users analyze the data for a different purpose by different applications. Therefore, data placement structure should be highly adaptive to meet the demand for dynamic data processing instead of fixed workload patterns.

Hive is a type of data warehouse system that supports a query language named HiveQL, which makes easy for end users to store and analysis large data set more intuitively that eventually meets the demand of data placement structure of Big Data. For

maintaining the data placement structure, Hive uses own file format to store the data set, which is RCFile and ORCFile. In this research, we conduct a study of performance analysis between RCFile and ORCFile formats of Hive system.

The goal of this work to find out the most useful file formats that satisfy the requirements of big data placement structure which are fast data loading, fast query processing, highly efficient storage space utilization, and strong adaptability to highly dynamic workload patterns. We have set up Hadoop cluster system with three nodes and made a series of experiments with the RCFile and ORCFile formats in terms of data loading, different query optimization and data storage space on Hive system. The result will find the efficient file formats for Big Data analysis on distribute system.

1.2 Organizations

The remainder of this thesis is organized as follows: the next chapter we describe the opportunity created by Big Data and cover the details of challenges, which we are facing at the time of processing and analyzing phase. These challenges are divided into two parts: the first part describes the phases of processing data and the second part discusses the difficulties of exploiting the data. In Chapter 3 reviews the Hadoop ecosystem in terms of Big Data solution. In Chapter 4 presents the data warehouse technology called Hive. In Chapter 5 introduces the RCFile and ORCFile as data storage structure. In Chapter 6 contains the detail presentation of performance evaluation of RCFile and ORCFile format in terms of data storage, data loading and query execution, and finally, Chapter 7 summarizes the results of implementation and future research work.

Chapter 2

Big Data Opportunity and Challenges

2.1 Big Data Opportunity

To grab the opportunity we need to glean information from Big Data. A study conducted by Bain and Company found that among 400 companies, those who have adopted Big Data analytics successfully have taken a leadership role in the corporate world. Scientific research, agriculture, urban planning, economy, energy, astronomy, healthcare, education, retailing, logistics, and business operations have benefited and revolutionized by Big Data (Bain.com 2013; Wall 2014).

Big Data Analytics is a wonder tool and According to the SAS.com magazine; the companies can get some benefits from Big Data Analytics as below (Davenport 2014):

- It is cost effective.
- It helps the organization to make faster and better decision.
- It provides an innovative way to create new products and services for companies.

The economic value of Big Data analysis is notable. In 2009, Google contributed 54 billion dollars to US economy for Big Data analysis and processes (Bertino et al. 2011). Another study shows that UK estimated that big Data analysis will be 322 billion USD dollars industry by 2017 and creates the opportunity for 58000 jobs for the nation (Brough 2013). Now a day a scientific researcher can find the natural resources by staying in their lab with the help of advanced sensing network. They can analyze the high-resolution seismic data which have been collected from the different geographical location for finding oil and gas resources. The Solan Digital Sky Survey becomes the significant central resource for an astronomer; its telescopes can collect pictures from the

sky and therefore, astronomers can find the interesting discoveries from their astronomical databases.

Proper Big Data analytics can boost the education sector (Dobbie and Fryer 2013). In New York City 35 charter schools have organized R&D to collect the proper data to improve school effectiveness. After analysis by different approaches, they can set up top five policies to improve school standards. Big Data effectively also has been implemented in election campaigns as well. For example, in 2008 and 2012 in both election campaign, Barack Obama had successfully used Big Data to control the voting processes and win the election(Rutledge 2013; Issenberg 2012). Obama campaign did approximately 50000 short time survey to people in every week to retrieve the voter information to reveal the preference about their next president. In the biological sector, scientists are now depositing real-time scientific data in a public database that is accessible by others scientist. The analysis of this updated data could yield drug discovery and curation of diseases worldwide.

2.2 Big Data Challenges

Big Data has a big value, and it is the next leader of innovation and productivity. The amount of created and stored data is almost unimaginable, and it keeps growing. The potential insight of it demands a relatively new approach to architecture, tools and practices. However, before taking the architectural design, the first and foremost task is to find out and make categories of challenges created by Big Data. The analysis of Big Data requires multiple phases, each of which introduces unique challenges (Bertino et al. 2011). Figure 2 shows the two-stage architectural view where left part describes the data

analysis pipeline and the right part is the challenging tasks that Big Data processing needs.

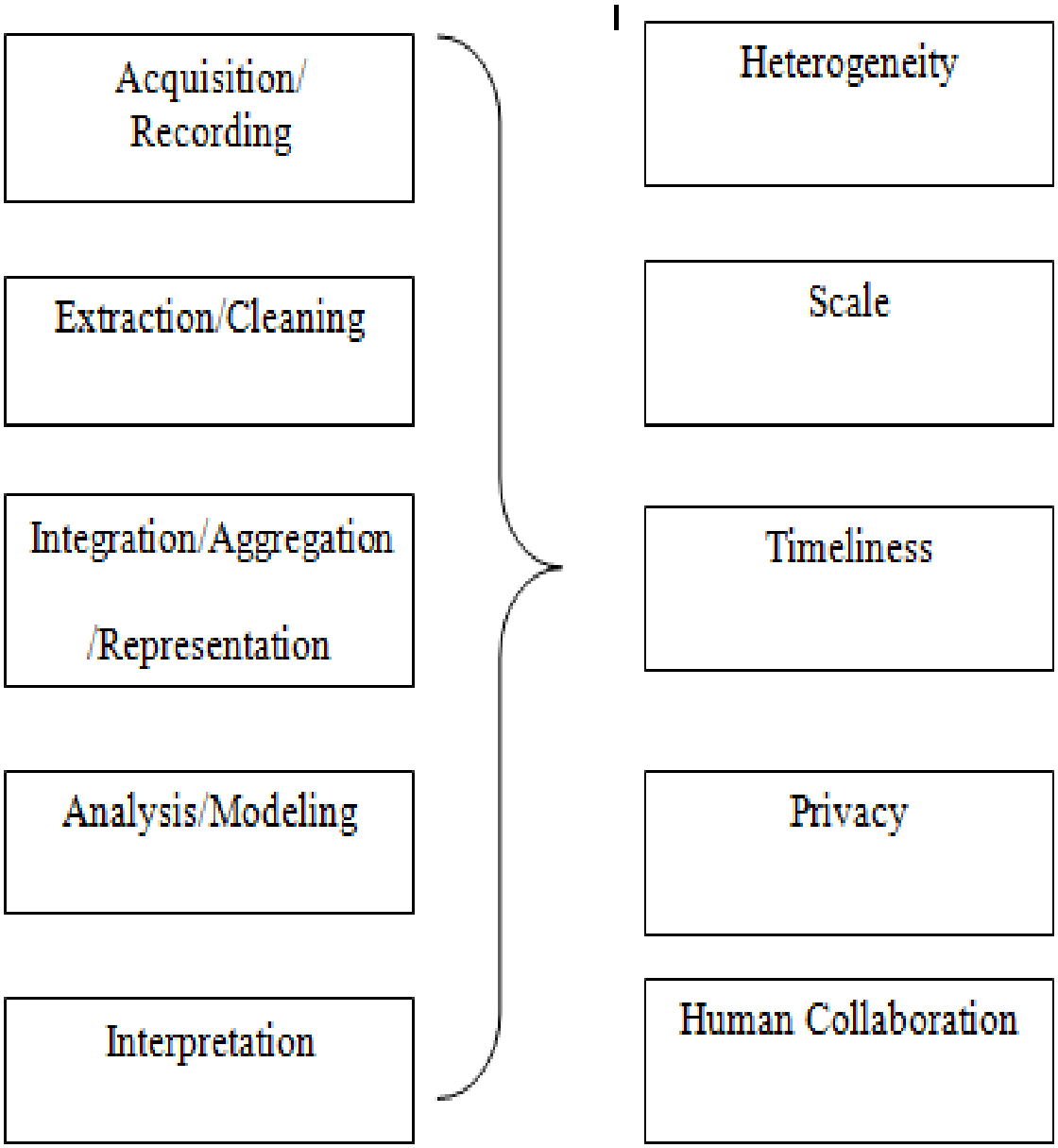


Figure 2. Big Data processing pipeline and challenging task.

2.2.1 Big Data Processing Pipeline Phases

2.2.1.1 Data Acquisition and Recording

Identifying the right data and determining the technique to best use is a major concern even though we can easily recognize the data generating sources. It is a very unwise decision to store and reduce the data at the same time because unnecessary data may need more time as well more space. We need to find out the technology to reduce the raw data both on-line and off-line while not missing the important information. The second big challenge of Big Data is to create metadata that helps to analysis the correct data in proper order. Data provenance is another important factor that identifies the subsequent processing of data with the origin and fit the data in the analysis pipeline.

2.2.1.2 Information Extraction and Cleaning

The collected amount of information are not immediately available for analysis, but it needs to bring out the correct format with correct shape of data since data are coming from different sources. The data can be structured, semi-structured and unstructured that all need to fit in a common method for analysis because different types of method exist for different sources of data. For example, medical reports of patients that contain structure information from sensors and measurement, and unstructured information from X-ray images. We need further extraction process to find the disease information from raw data of medical reports. It is continuing the resolution of technical challenges in a correct and comprehensive way to make different raw data types more suitable for analysis.

2.2.1.3 Data Integration, Aggregation and Representation

Data integration phase makes available multiple data sources for access where sources can be virtual like link direction or warehouse-like easy query (Brough 2013). Big Data are generating at an unprecedented speed from various sources with different types of structures. It is not enough that this heterogeneity of data just holds and store in the warehouse system for analysis. Since we know, the data are coming with different structure so before storing it into the database, we need a good design of data arranging system that helps for finding the relevant information at analysis time. Big Data analysis challenges are not only the warehouse problem and finding the techniques to analysis the information but also making decision automatically with proper design and maintain a good data structure.

2.2.1.4 Query Processing, Data Modeling and Analysis

Data Mining and query processing of Big Data are very different from database management systems. In traditional database systems, data are stored in their relative field in a structured way, but in Big Data there is no fixed structure and also sources are different and uncounted. Big Data is noisy, dynamic, heterogeneous and untrustworthy. Noisy and untrustworthy data are sometimes more powerful than structured data because data are interconnected through a network which can have the knowledge of missing data and finally helps to find out hidden and trustworthy relationship and model (Bertino et al. 2011). Data mining helps to improve the quality of noisy data to understand the semantics and set up of a model for intelligent querying functions. It also helps to develop a knowledge-base method from interrelated data to make better assumptions.

Real-time query processing for the content creation web-site will be effective in the future to decide whether to store or discard it. Query processing technique from the vast amount of data in certain time is remaining a hectic research problem today.

2.2.1.5 Interpretation

Interpretation carries the meaningful result and explains the data architecture that helps to analysis for the decision maker. Without understanding the hidden insight, Big Data are meaningless. There are many reasons why interpretation is important since assumptions based on different criteria might be wrong, or the computer systems may have some bugs or the results are made based on wrong formats (Bertino et al. 2011).

Interpretation does not refer to bring out the results with nice presentation view, rather than it can help to make fruitful decisions from multiple phases of results. The early business sectors were depending only on tabular representations. Modern companies are not satisfied only with visual representation but also need the support for user collaboration and all supporting criteria for interpretation.

There are different kinds of analysis are available that these days companies use to make a decision for further analysis such as web analysis, mobile analysis, marketing analysis and predictive analysis (Hoovers 2013). These might be helpful for counting visitors and their behavior of certain pages, from where they are coming which help to provide advertise, help business by segmenting account by different criteria such as geographical location, industry, a size which can give a better idea of similar prospects.

2.2.2 Challenging Task of Data Processing

2.2.2.1 Heterogeneity and Incompleteness

The Big Data is mainly machine-generated data, and the big question is how fast and how much data we are storing in our database. When data are generating by human factor heterogeneity and incompleteness can effectively handle by users. However, it is in a different case for machine analysis algorithms, which expect homogeneous data and need to analysis to convert structural format before going extract information (Bertino et al. 2011).For example, we can think a medical report for the patient. If we want to store the disease history information for a patient and it would difficult to find any pattern of data because there are no predefined information and further it might be different for every report. Incompleteness is another issue, which can hinder the analysis of Big Data. Even after removing the heterogeneity there may be still errors due to lack of information.

2.2.2.2 Scale

Big Data requires three criteria for scaling, which are data volume, hardware size and concurrency (Zicari 2011). The size of data is the most challenging issue because we cannot predict the amount of data is going to be stored in the databases. In the past, faster processors were used to manage a large volume of growing data sets. Due to power constraints, the processors are being built with increasing numbers of cores rather than doubling the clock cycle frequency of processors (Bertino et al. 2011).

Furthermore, parallel clusters system can be very effective to handle the increasing volume of data because the hardware resources are shared processors cores. In the cloud computing, scaling system plays an important role to integrate the multiple clusters' workloads and gives the concurrency support to prevent system. However, despite discovering many techniques, Big Data scaling is still in challenging tasks, which demands to rethink the existing designs, builds and data processing components.

2.2.2.3 Timeliness

Processing large data sets needs longer time. Therefore, effective design of a system makes processing of these data sets faster. Timeliness is not just only refers to the velocity of data, it also closely related to acquisition rate or currency of data (Bertino et al. 2011). We have to make sure that most updated data are stored in the warehouse despite the modifications and changes the data over time. In the current age, most of the data are web generated and it is extremely needed that data driven should be in real time (Harris 2015). For example, credit or debit card companies handle millions of data in every day, and it is their responsibility to give the full protection of the users that there are no suspicious activities on the account. If any fraudulent or unauthentic user's activities have been traced, they should take action immediately about the potential threats and stop the next transactions. These kinds of operation should be completed in a very short time and real time (Bertino et al. 2011).

Sometimes, the real-time decision is very important to meet the specific criteria. For example, Google map services which inform the users of possible traffic congestions and therefore immediately calculate the available fastest route. To create a decision for

such moving objects in real time from millions of data is a big challenge when data are generated quickly and need a tight response time of availability of users.

2.2.2.4 Privacy

To keep information safe and secret are a challenging task in the context t of Big Data. Storing, scaling and providing timely services are good enough until the information is safe. Personal data with linking multiple data sources are most dangerous for both technical and sociological problems. For example, location-based services (Bertino et al. 2011) which are a very popular application of smartphone devices helps people to find their desired locations despite the user is totally unknown about their present locations. Hiding users' identity is not possible, and a culprit can easily infer the users' information. For example, an attacker can track the users' all connections where he traveled over the time, and it is easy to figure the user's private information such health issue, religious matter, job information and so on. This era is called Internet explosion era, where people love to visit and share their information in social networking, and it is impossible to hide the users' details (though all information is saved in secured database). Because all the users' profiles are visible to everyone and the hacker does no need to hack the main database to get the desired data.

2. 2. 2.5 Human Collaboration

Despite all the automation systems, the machine needs human collaboration to overcome the Big Data analysis successfully. Knowledge based method is necessary for data preparation or data assembly which depends on data quality and documentations. Without knowledge-based method, the resulting analysis might be erroneous. Everybody

should have a different role in solving Big Data analysis such as a computer programmer able to write code; a data scientist can discover good algorithm and machine finally can find out hidden patterns (Rockwell 2015). However, these codes, algorithms, and patterns are meaningless if there are false predictions. For example, In 2015, UK regulator fined for wrong bill statement and they clarified that it is the software coding errors (Rockwell 2015). These types of mistakes can be handled if there is a co-equal and co-dependent collaboration of man and computer (Kobielus 2014).

Chapter 3

Hadoop

3.1 Hadoop Overview

Hadoop is a Java based open source software framework developed by Apache Software Foundation, which can store, process, and analyzes a large volume of datasets in parallel on clusters of computers. Hadoop can scale up easily from a single server to hundred servers, where each server provides local computation and storage capability. The Hadoop cluster plays an important role in storing huge amount of data for large companies around the world. For example, Yahoo has the biggest Hadoop cluster of 4500 nodes that stores 455 Petabytes of data, where Facebook has 2000 nodes with more than 21 Petabytes of data in a single HDFS cluster (Asay 2014; Won 2016; Dhruba Borthakur 2010; JoshBaer 2015).

3.2 Hadoop Architecture

Hadoop is a software of libraries, not a single module, which provides an ecosystem, where different parts of Hadoop components have a specific function to do and integrate with data processing technologies(Hortonworks 2014). The Hadoop library is also able to detect and handle the hardware failures at the application layer (White 2015; Wikipedia 2016a).Hadoop has four core modules as shown in figure 3 (Tutorialspoint 2015a; Dezyre 2015).

- Hadoop MapReduce: The programming model for parallel processing of large data sets.
- Hadoop Distributed File System(HDFS): The default reliable and scalable

file system to store a large volume of data in the clusters.

- Hadoop Yarn: The Framework to adjust the workload capabilities and dynamic management of job scheduling.
- Hadoop Common Utilities: Contains the collections of common libraries and utilities that support other components of the Hadoop ecosystem.

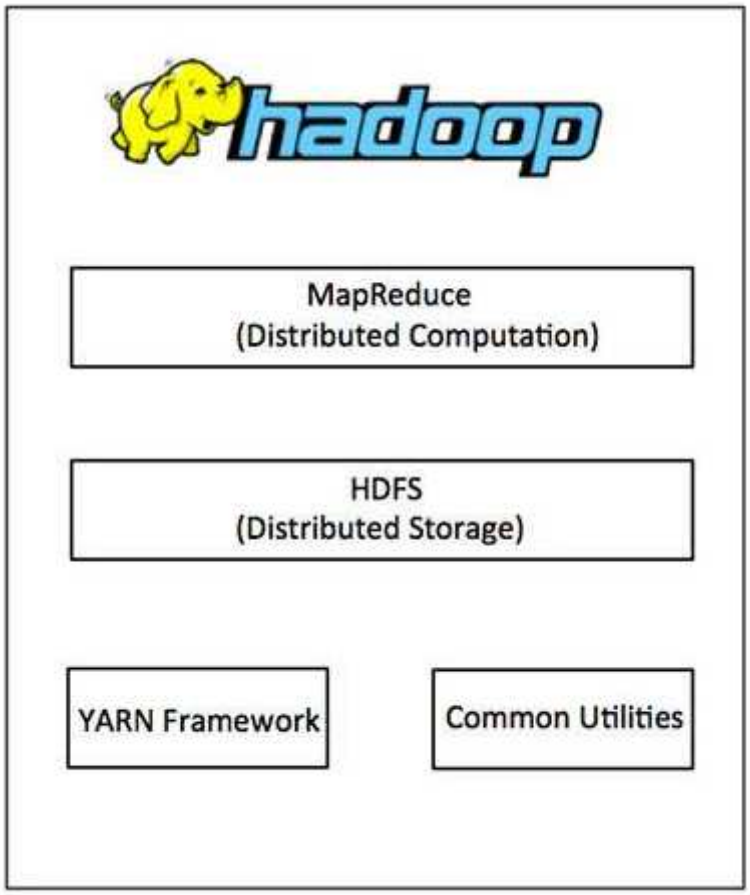


Figure 3. Hadoop core component module.

3.3 MapReduce

MapReduce is a programming model to process a large volume of datasets in parallel on clusters of computers (Tutorialspoint 2015b).The MapReduce has two methods: Map and Reduce. The Map method takes the raw data as input and breaks the data into numbers of smaller data sets. Each data set in a Map method receives a key/value pair and produces intermediate key/value pairs, and stores the output of a temporary storage system for further processing. The Reduce method combines all the intermediate key/value pairs based on the intermediate key and generates new sets of output. Besides Map and Reduce methods, shuffling is another process to transfer the data from Map processes to Reducers. The MapReduce tasks happen in the local disk to avoid the network congestions, and the results will be sent to the appropriate servers (White 2015). Figure 4 shows the architecture and data flow of Map--Reduce tasks (Khan et al. 2014).

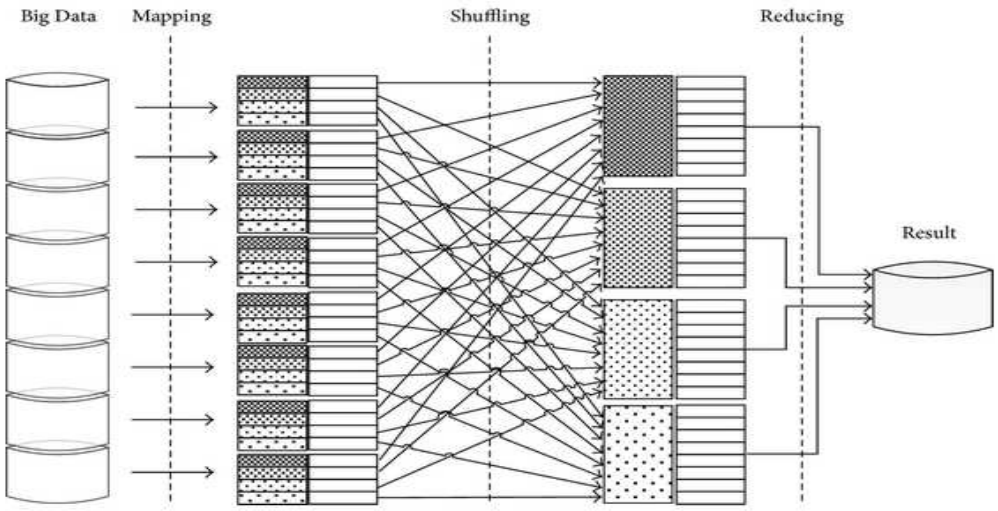


Figure 4. Data flow and MapReduce architecture view of Big Data processing.

3.4 Hadoop Distributed File System (HDFS)

A file system that handles the storage capacity of the network is called the distributed file systems. Hadoop introduces its file system named Hadoop Distributed File System (HDFS). It makes partitions across the number of separate machines, which provides significant advantages when the storage capacity of the single machine is not enough to store large datasets (White 2015). The HDFS goals for Big Data processing are as follow (D Borthakur 2008):

- Detect hardware failure from hundred to thousand of machines and ensures automatic recovery.
- Provide streaming accesses with high throughput to avoid low latency data access.
- To support write-once-read-many access model. If a file created, written, and closed need not be changed in HFDS, which is helpful to append to the files when necessary.
- To allow computation of data, where data is located to minimize the network congestion and increase the overall throughput performances.
- To facilitate the design of portable hardware and software platforms, which provides the widespread adoption of large set of applications.

3.4.1 HDFS Architecture

3.4.1.1 NameNode

The HDFS cluster has master-slave architecture, which introduces master as NameNode, and slave as the number of DataNodes(White 2015). NameNode controls the file system namespace and stores the metadata across the clusters. Metadata contains

the information about where and which DataNode have stored the data files. The data files are break into multiple pieces of blocks. The size of each block is 128 MB by default and stores in a set of DataNodes.

NameNodes are responsible for namespace operation such as opening, closing, renaming files directories, and mapping blocks to DataNodes("HDFS Architecture" 2015). It does not control block operation since DataNodes arrange the block whenever the system starts. Hadoop provides two mechanisms to make a NameNode consistent and protect it from the single point of failure. The first one is creating backup files of metadata to multiple file systems, and the other one maintains a secondary NameNode in a different machine. The secondary NameNode periodically merge the namespace images and keeps an updated copy in its own spaces. It provides a backup NameNode when original NameNode fails. Figure 5 shows the architecture of HDFS.

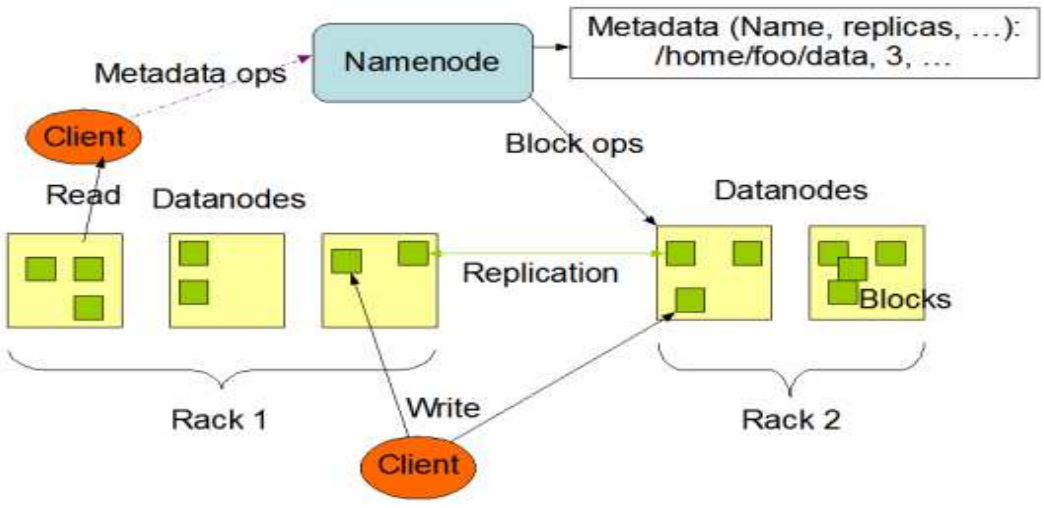


Figure 5. Hadoop distributed file system architecture

3.4.1.2 DataNode

A DataNode stores data in different blocks in HDFS and allows read /write operations by clients ("HDFS Architecture" 2015). It is also responsible for performing block creation, deletion and replication according to the instructions by NameNode. All the DataNodes and NameNodes are communicating by heartbeat messages every 3 seconds. A DataNode is considered to be dead if it does not receive a message within 10 seconds. The DataNodes communicate to each other to move, copy, and replicate and calculate the checksum of the data (Jain 2012).

3.4.1.3 Data Flow to Read and Write from HDFS

To read the data from HDFS, the client calls the "open ()" method on the FileSystem object. Then, DistributedFileSystem contacts with the NameNode using remote method invocation (RPC) to determine the files location. The NameNode provides DataNode location and FSDataInputstream to help with finding the files inside the DataNode. If any error occurs while reading, FSDataInputstream finds the nearest blocks of data. Client closes the connections by "close ()" method on FSDataInputstream when it finishes the "read" operations. Figure 6 shows the data flow to read data in HDFS (White 2015).

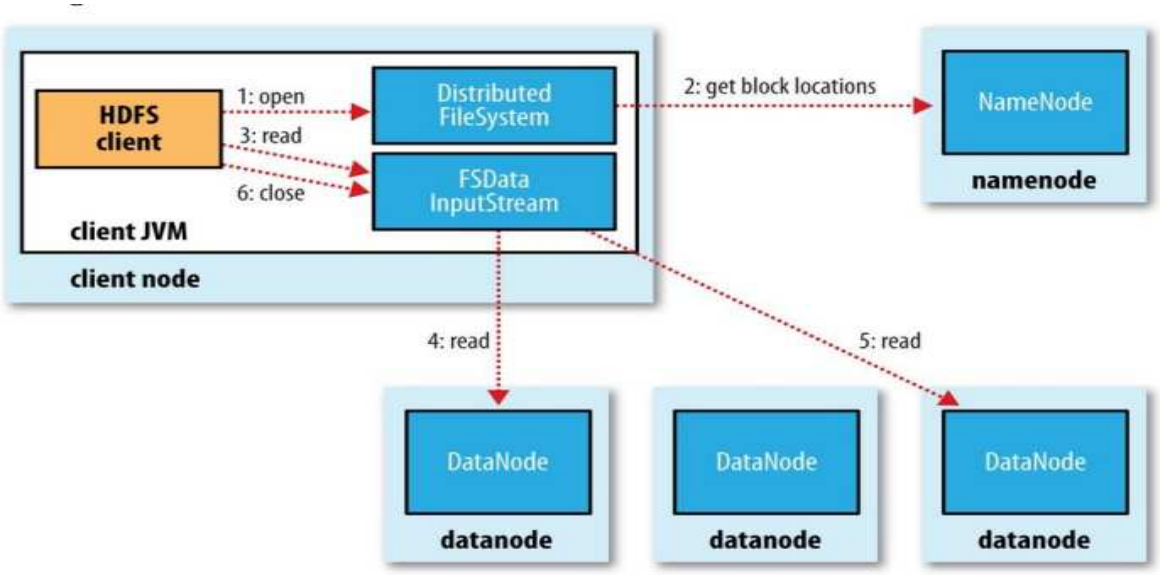


Figure 6. Client reading data from HDFS.

To write the data in HDFS, clients creates a file by write () method to DistributedFileSystem, which invokes the remote method (RPC) to call the NameNode for creating new file system's namespace. The NameNode checks the file existences and the user's permission for creating a record for new files. DFSOutputstream splits data into packets and includes the queue locations. The DataStreamer asks the NameNode to allocate new blocks and the list of DataNode for replications. When all the DataNodes send acknowledge to the NameNode, then replication is complete, and a packet will be removed from the ack queue. After writing has finished, client calls “close ()” method and informs the NameNode that the file has already written into the DataNode. Figure 7 shows the steps for writing data into HDFS system (White 2015).

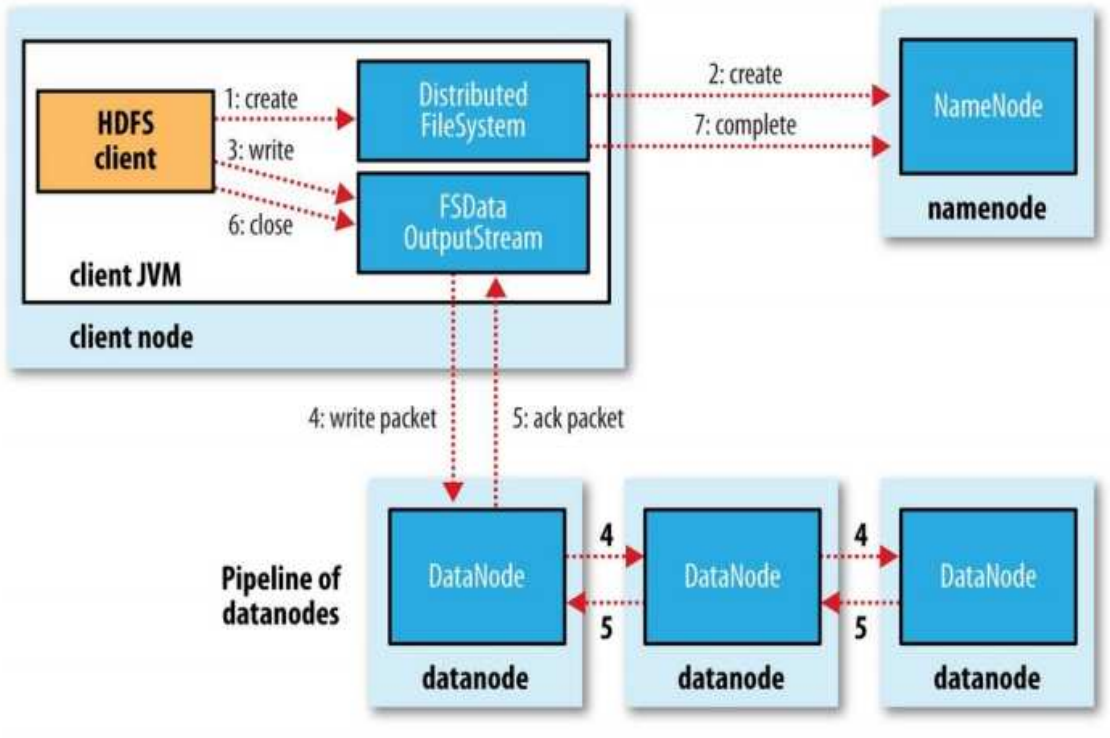


Figure 7. Client writes data to HDFS.

3.4.1.4 Data Replication

In HDFS, the size of all blocks is same except the last block and users can define the size and the number of blocks. The each file is replicated for supporting the fault tolerance of HDFS. The NameNode is the master for replication of blocks and receives periodic messages from DataNode to ensure that all blocks are working properly. The placement of replication is very important to improve the reliability and performance of write and read bandwidth in HDFS. In each DataNode, the rack id is maintained by NameNode and the NameNode places replicas in the unique rack to prevent data loss when entire racks fail. The number of default replication factor is three; the first a replication put in one local rack, second is it puts on a node in the different remote rack

and the last it puts on a different node but in the same rank. Figure 8 shows the block replication of NameNode and DataNode in HDFS.

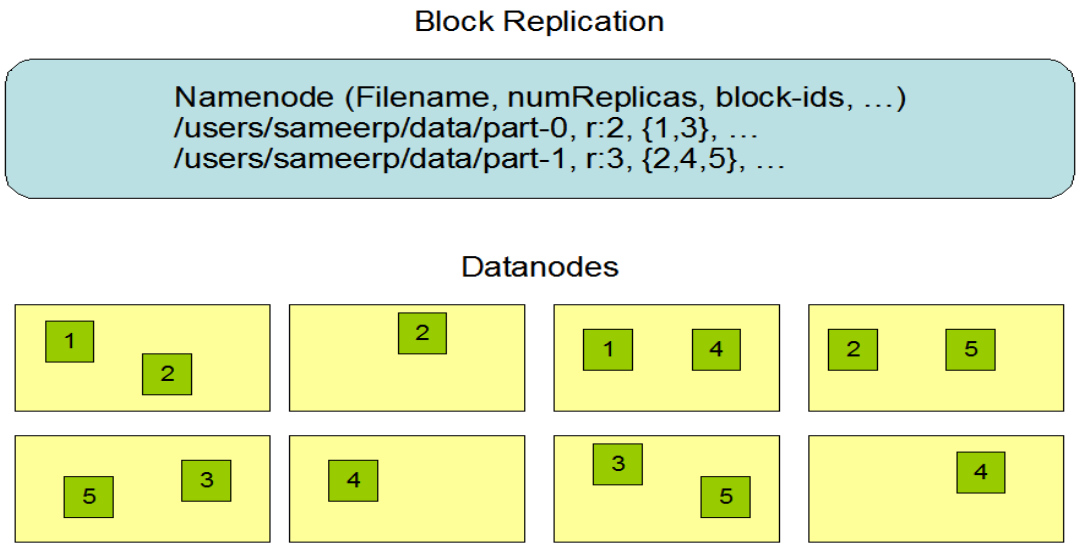


Figure 8. Data replication in HDFS block.

3.5 Hadoop Yarn

The Hadoop Yet another Resources Negotiator (YARN) is a cluster resource management system for job scheduling to improve the MapReduce implementation in Hadoop. MapReduce is not effective alone to process large data sets, where YARN initiates multiple MapReduce jobs for one application (Murty 2012).MapReduce, Spark, Tez are the examples of YARN applications and clusters storage system(HDFS and HBase) (White 2015).

YARN consists of ResourceManager, which controls the use of resources across the cluster and a NodeManager, which takes command from ResourceManager to lunch and monitor containers. The container provides a set of resources to run an application

(Memory, CPU and so on). Figure 9 describes the Yarn architecture and the processes to run an application.

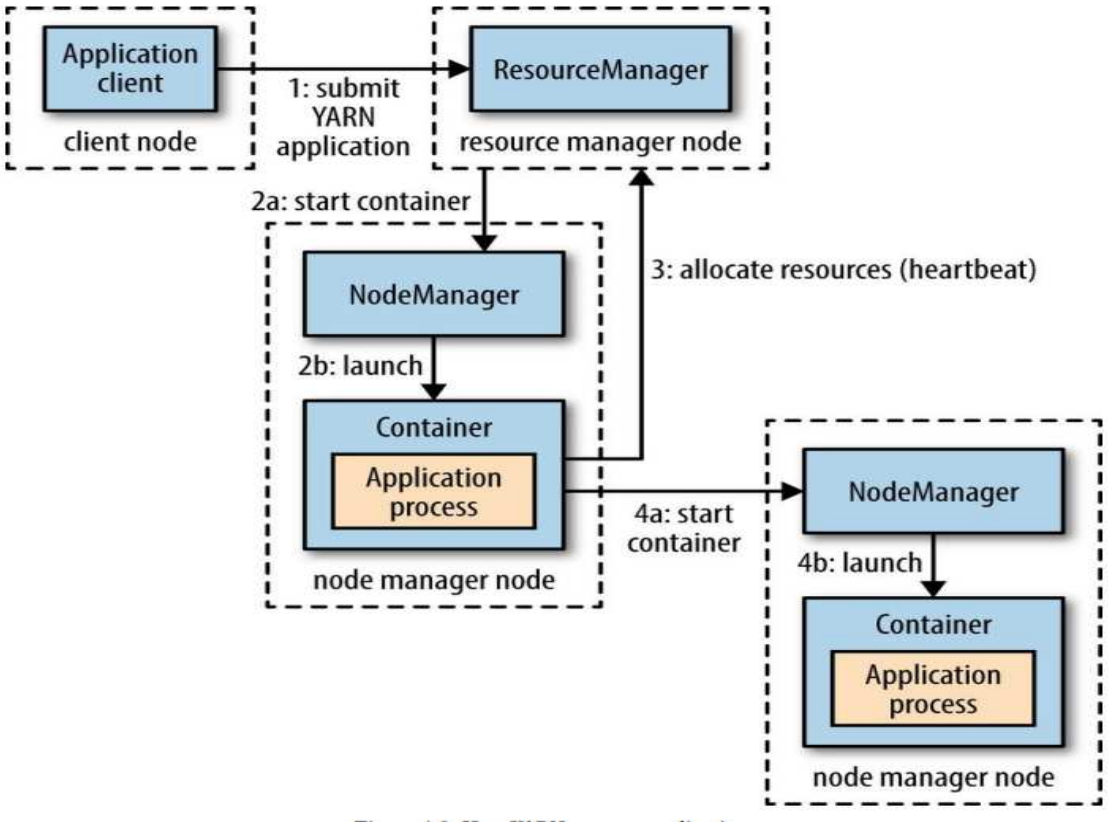


Figure 9. Architecture view of YARN to run an application.

To start a YARN application in distributed Hadoop clusters, a client has to follow several steps (Murty 2012). First, a client node needs to contact with ResourceManager by submitting a YARN application including all jobs specification and request to run an ApplicationMaster process. ApplicationMaster negotiates with ResourceManager for resources and monitors the containers for resource utilizations. The ApplicationMaster contacts with ResourceManager to communicate directly and executes the computation

within containers. After the application is complete, ApplicationMaster de-resisters with ResourceManager and shut down to allow the container to be repurposed.

3.6 Hadoop Common

Hadoop Common has some predefined libraries and utilities, which are useful for specific data I/O operations (Apache Software Framework 2016). The native Hadoop library includes various types of components for data integrity, data compression and checksum computation. For data integrity, Hadoop uses common error detecting code called CRC-32, which computes the checksum of the input files. For file compression, Hadoop common provides different compression formats, tools, and algorithms. Serialization and de-serialization are important for turning the structure object to byte streams and byte streams back to structure objects. File-based data structures such as sequence and map files are also common in native libraries for Hadoop (White 2015).

Chapter 4

Big Data Warehouse

4.1 Hive

Apache Hive was developed by Facebook infrastructure team to manage the growing volume of data sets that Facebook produces everyday from its social networking activities (White 2015). Hive has included in Hadoop ecosystem as a subproject, which has been used as a data warehouse framework. Hive supports SQL like scripts called HiveQL, to run a query on a large volume of data sets using MapReduce. Hive maintains metastore to contain schema and statistics for data exploration, query optimization and query compilation. In Hive, data are organized in three formats, which are tables, partitions, and buckets. The tables are much like to the relational databases system.

The data inside tables are serialized and stored in HDFS directory. Hive provides built-in serialization, de-serialization and also users' define custom serialize and de-serialize method. Hive supports all primitive and complex types of data such as bigint, int, smallint, tiniint, float, double and object. The second format is the partitions, which divides the data tables into subdirectories, which are defined by data type characteristics. The last of data format is buckets, which can store data in both partitions and table's directory that depends on whether the table is partitioned or not (Thusoo, Sarma, and Jain 2010; Tutorialspoint 2015c).

4.2 Hive Architecture

Figure 10 shows the system architecture and components of Apache Hive system (Thusoo, Sarma, and Jain 2010). The Hive component contains different units, which are the following:

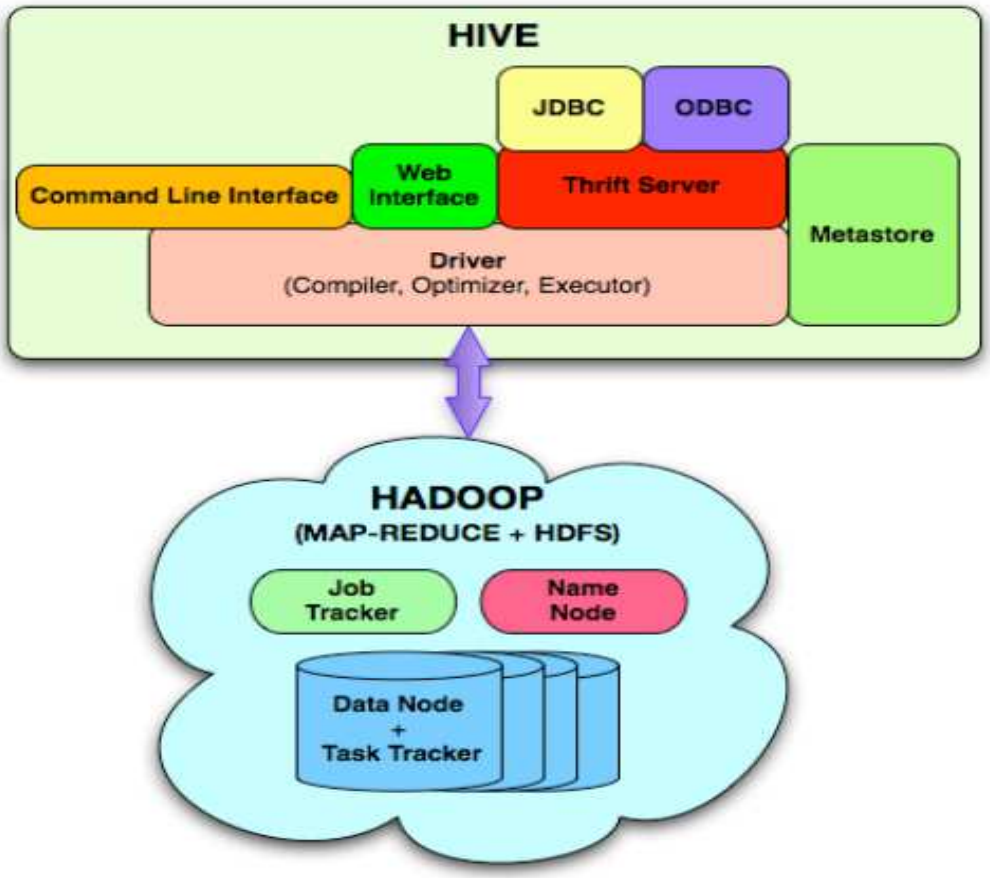


Figure 10. Hive System architecture and components.

- External Interfaces: Hive supports different types of interfaces to initiate the works between user and HDFS, such as Command Line Interfaces (CLI), Web Interfaces, and programming interfaces (JDBC, ODBC).

- Thrift Server: The Hive thrift server supports cross-language services, which works with clients API to execute query statement. Thrift server works with common drivers such as JDBC/ODBC driver and also provides a platform to integrate other applications with Hive.
- Metastore: Metastore helps the Hive to store the system catalog and metadata, which contains details information about tables, columns, and partitions and so on. (Thusoo, Sarma, and Jain 2010)
- Driver: Driver maintains sessions and statistics of HiveQL statement, which moves to Hadoop through Hive.
- Query Compiler: The component that compiles user defined HiveQL into MapReduce task.
- Execution Engine: Execution engine is responsible for executing the task produced by compiler and MapReduce, which maintains the dependency order to communicate with the Hadoop modules.

4.3 Workflow Between Hive and Hadoop

Figure 11 shows the step by step workflow between Hive and Hadoop components. In the first step, the interface sends a query to the database driver for execution. The driver takes the users' jobs and informs compiler to parse the query syntax. The compiler sends metadata to the metastore and metastore sends back metadata to the compiler (Tutorialspoint 2015c). Metastore selects the data storage systems and compiler checks the necessary requirements to complete the query. The execution

engines (MapReduce) sends the jobs to NameNode, which assigns DataNode for query executions. Finally, execution engine gathers all results from DataNodes.

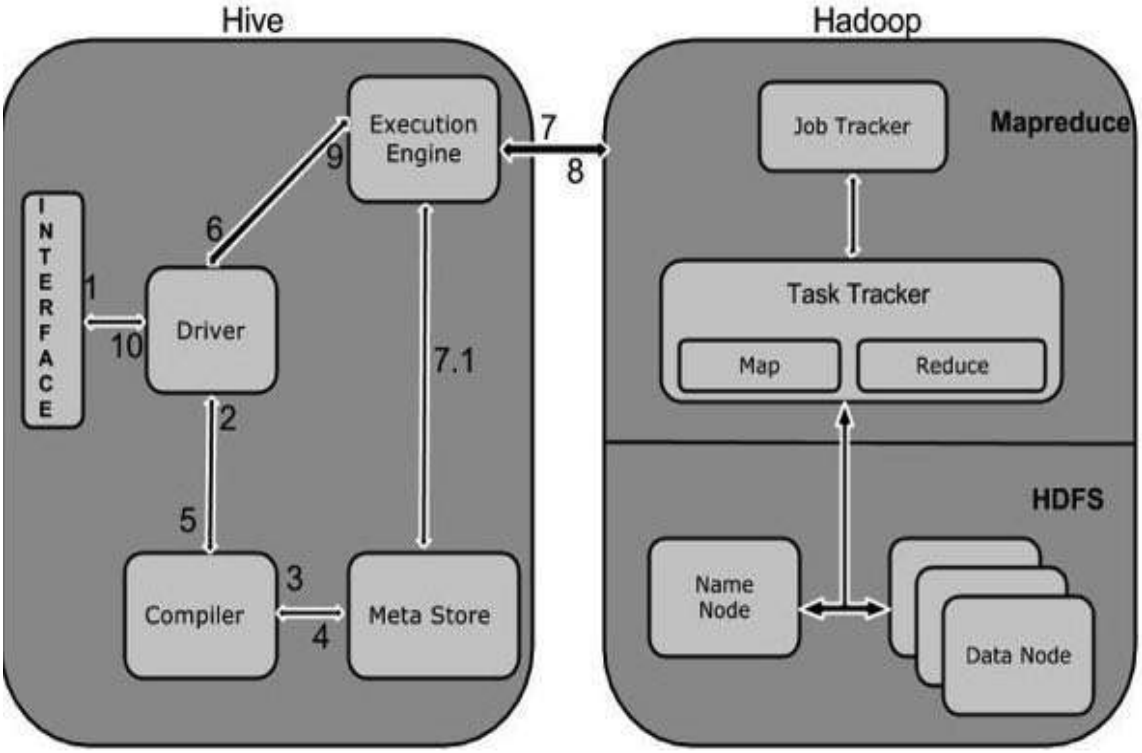


Figure 11. Hive working flow with Hadoop component

Chapter 5

Data Placement Structure

5.1 Record Columnar File (RCFile)

RCFile (Record Columnar File) is a data storage format for MapReduce-based data warehouse systems (HIVE) to organize and store a large volume of relational tables data in HDFS clusters, which is a combination of multiple features such as data storage format, data compression and data optimization techniques (Datametica 2014). In this data storage format, tables are stored first in horizontally, and then vertically to organize each column independently in clusters. RCFile supports column-wise data compression technique (lazy decompression) within each row group that helps to avoid unnecessary column reads during query execution. Furthermore, RCFile allows to select flexible row group size and arrange the same row data in the same node that increases the performance of data compression and query execution (He et al. 2011).

5.1.1 RCFile Data Architecture

In HDFS, all the row group sizes are the same and a table can contain multiple blocks and organizes data in row group way. A block can have one or multiple row groups based on the size of row group and the HDFS block size (White 2015). Figure 12 describes and illustrates the RCFile architecture (He et al. 2011).

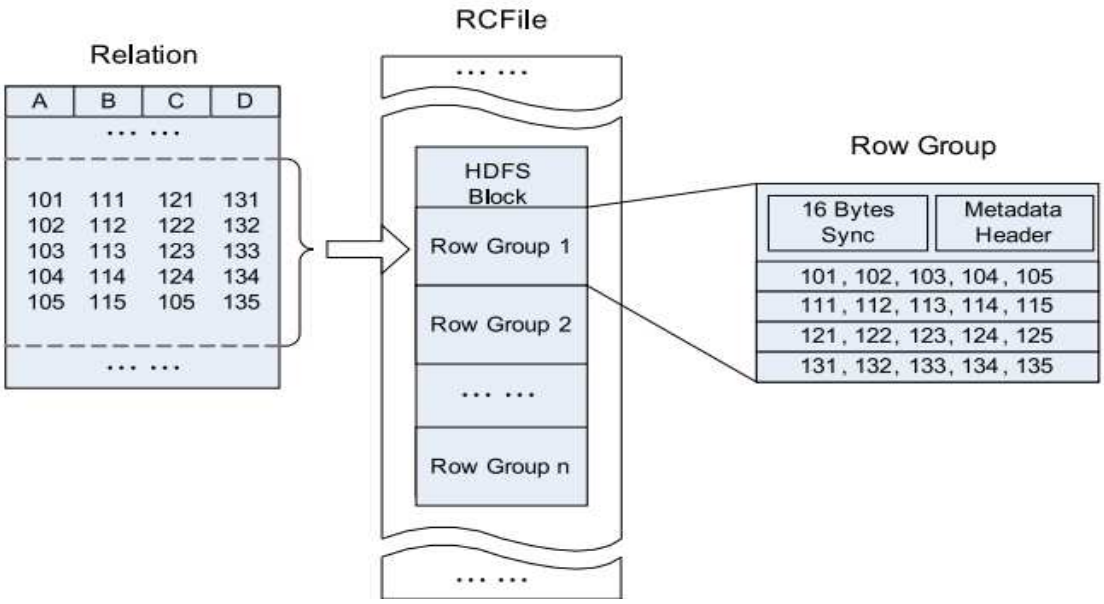


Figure 12. RCFile layout structure.

A row group has three components, which are synch marker, metadata header and column store. Synch marker is the beginning placement and helps to isolate the two-row groups in an HDFS block. Metadata stores the information of all row groups, such as how many row groups are placed, the size of each column and as well as the size of each field in a column. The last one is column store, which helps to arrange all the fields in the same column together.

5.1.2 RCFile Data Compression

In RCFile, metadata header and table data sections are compressed separately. The metadata header section uses RLE (Run Length Encoding) algorithm to compress as a whole unit and the data table section compress each column independently (He et al. 2011).

The RLE can find long run repeated values because all the values in a column are stored in continuously. RCFile uses GZip algorithm for better compression because it only compresses data when all columns are sorted. For each column, RCFile supports separate algorithm to compress data in the table section. RCFile does not support random writing operation rather than it provides an interface for appending the end of the file. RCFile maintains a memory column holder for writing data in each column. Before writing data to the disks, RCFile uses two parameters to control the memory buffer. The first parameter is to control the number of records, and the second parameter is to control the size of the memory buffer.

5.1.3 RCFile Lazy Decompression

In MapReduce framework for each HDFS block, a mapper is worked sequentially to process each row group. For reading, RCFile does not read the whole file rather than it reads only metadata and the corresponding columns to avoid the reading of unnecessary columns in the row group. Then the metadata header and compressed columns are loaded into the memory for decompression. RCFile uses lazy decompression technique that remains in memory until the other row groups are processed. Lazy decompression becomes useful when there is a where condition in a query, because if some row groups do not satisfy the where condition then those row groups do not need to be decompressed. For example, consider a table (T) with the following columns (col1, col2, col3, col4,col5, ….) and there is a query such as select col1, col3 from T where col2 = 2. The RCFile only reads the metadata header in the row group and decompresses only those row groups

that match the where condition (col2=2) of the above query, not other row groups that do not match the where condition, which saves time.

5.2 Optimize Record Columnar File (ORCFile)

5.2.1 ORCFile Structure

Like RCFile, ORCFile is an another basic data placement structure to store and organize relational data in Hive. ORCFile maintains one file for collections of rows, which is arranged in a columnar format that allows parallel processing of row collections in clusters (Prokopp 2014; He et al. 2011). An ORCFile structure has three parts; these are stripes, footer and postscripts. Figure 13 shows the file structure of ORCFile (Leverenz 2015).

Stripes hold the groups of row data and footer maintains a list of stripes in the file, which contains the information of a number of rows per stripe and column's data type that includes aggregate functions such as count, min, max and sum. Each stripe size is 256 MB by default, which is good for a sequential read on HDFS. The larger block size reduces the load of NameNode because the users can read more data from a single file.The last one is postscripts, which maintains compression parameter and the size of the compressed footer. The stripes have divided further into three parts, which are index data, row data and stripe footer. Index data maintains the information of min, max values and the row positions for each column. These row positions are very efficient to find the specific compression and decompression blocks by providing the offset of indexes, where indexes are used to select the stripes and row groups. The row data stores multiple

streams of per column independently and uses them for table scans. Stripe footer provides directory services such as encoding types and stream locations.

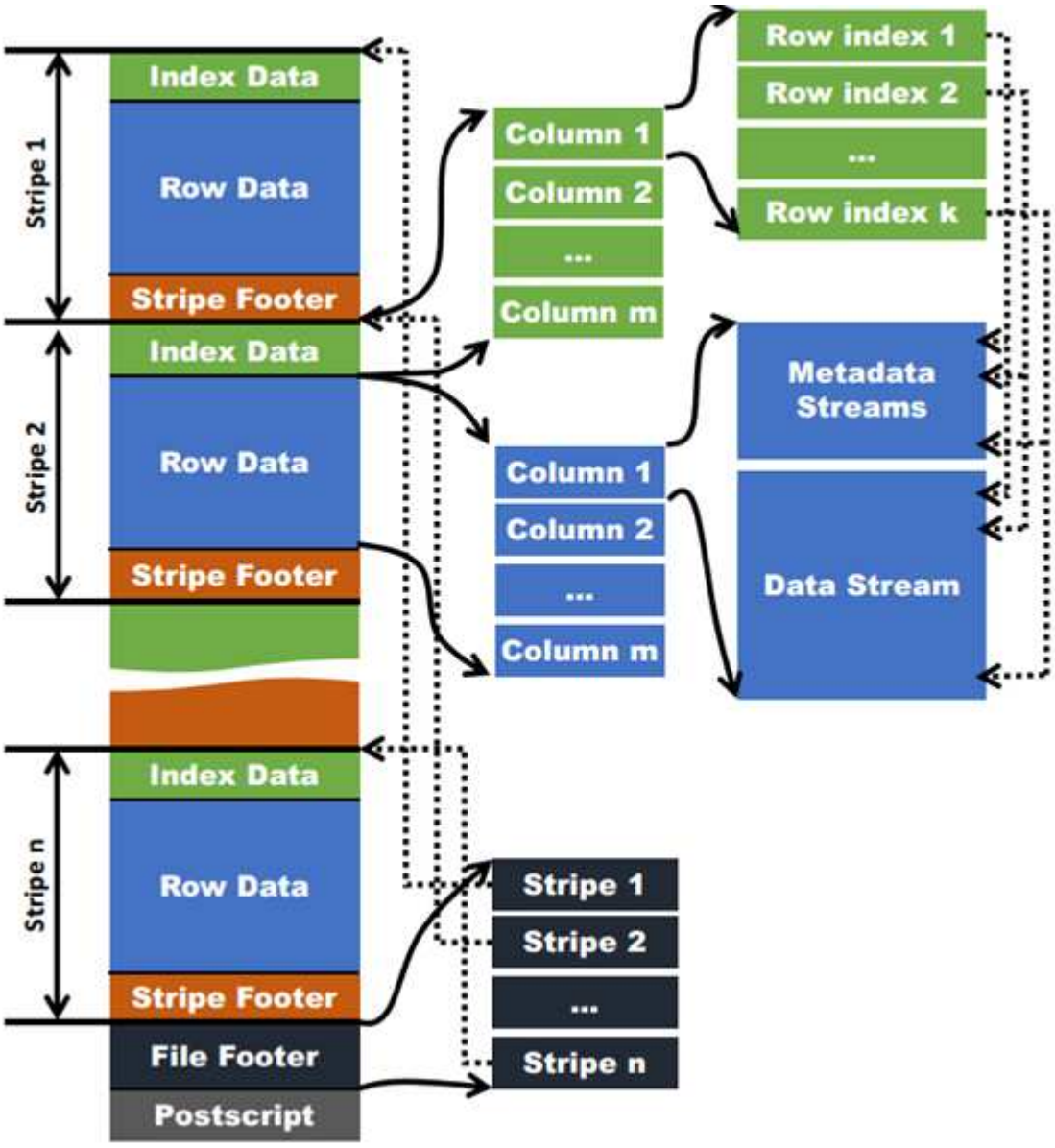


Figure 13. ORCFile structure.

5.2.2 Data Write and Compression

The ORCFile writer does not shrink the tables or whole stripes at a time rather than it applies data encoding and compression techniques (Huai et al. 2013). To write data into HDFS, ORC uses memory manager to buffer the whole stripe in memory. Due to large stripe size, ORCFile uses multiple writers concurrently to write data in a single MapReduce task, where memory manager controls the memory consumption of writers (Huai et al. 2014).

It supports two types of compression techniques. The first one is automatically used as type-specific encoding methods for columns with various data types and the second one is optional compression codecs set by users called generic compression. A column in a stripe can contain multiple streams, where each stream can be divided into four primitive types. The primitive streams are byte, run length byte, integer, and bit field streams. Each stream uses the own encoding technique, which depends on streams' data types. For example, integer columns data type are serialized into two streams, which are present bit stream and data stream. For one bit or small integers, the variable length encoding is used, and for the long data streams of integers, the run length encoding technique is used. Besides using these type-specific encoding, users can also compress an ORCFile by general purpose codecs such as ZLIB, Snappy, LZO (Leverenz 2015).

5.2.3 Data Read, Lazy Decompression and Lazy Decoding

In an ORCFile, the performance of data read is enhanced by lazy decompression technique (Vagata and Wilfong 2014). Without lazy decompression and lazy decoding, a query seeks all the stripes to bring out a specific column, which will take a long time to

finish the MapReduce tasks. This decoding technique is used index stride that already existed in ORCFile format. The index stride helps the reader to skip unnecessary stripes and only decompress and decodes the target columns needed by the query. The footer in the ORCFile has all the stripes that contain the streams location. Thus, the users query only read the stripe lists to find the appropriate stripe location.

Chapter 6

Performance Evaluation of RCFile and ORCFile

6.1 Overview

In this chapter, we have conducted a series of experiments based on different criteria to measure the efficiency of RCFile and ORCFile format. Our experimental results will guide to choose the best file format for data placement structure in Hadoop MapReduce-based warehouse system. For this experiment, first we will introduce the environment setup, and then present performance results of the above two file formats.

6.2 Experimental Setup

The effectiveness of distributed systems depends on how one can perform the read and write operations, where the format of stored data is an important metric for completing these operations successfully. In this thesis, we choose three aspects to determine the best file format between RCFile and ORCFile, which are data storage space, data loading time, and query execution time. For this work, we have configured a virtual Hadoop cluster in the lab machine, which consists of three nodes. The cluster works as master-slave architecture, where master node maintains the workspace to distribute, store and replicate data to slave nodes. Figure 14 shows the system architecture and network configurations of our experiments. We have installed virtual box software in each machine to configure Hadoop environment. The operating system in each virtual box was Ubuntu 14.04.2 LTS 64-bit. The host windows machine has 8 GB memory with 3 GHz Intel Core i7 CPU, where each node in the virtual box has shared 8 GB memory with 60 GB disk.

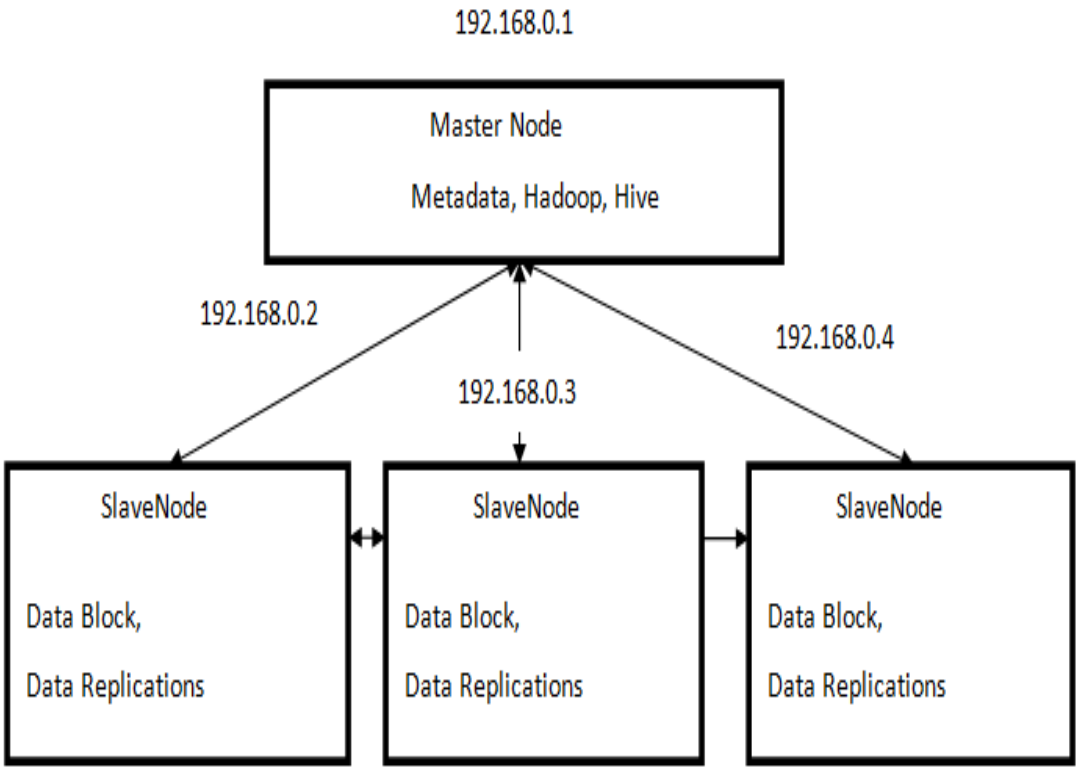


Figure 14. System architecture and network configuration of experiments.

In our experiments, we have used Hadoop-2.7.1 and for data warehouse, we choose Hive-1.2.1. We have chosen MapReduce as an execution engine since it is the default data processing engine used by Hive. The HDFS block size has set to 128 MB, and the replication factor is three. For these experiments, we have used ~6 GB dataset, which consists of movie reviews from Amazon. This dataset contains more than 5 million reviews including eight columns.

We have divided our experiments into two parts. In one part, we have compared the RCFile and ORCFile without using any compression algorithm. In the second part, we have applied compression algorithm (LZ4) to evaluate the performance of both

RCFile and ORCFile from the perspective of data storage space, data loading and query execution time. Table 1 shows an example of experimental dataset data type and data information

Table 1. Details information about data types and example of data set

Column Name	Data Type	Example Data
Productid	String	B00006HAXW
Userid	String	A1G69BQLIUMWPN
Profilename	String	cassidrm
Helpfulness	String	3/4
Score	String	5.0
Time	String	1200096000
Summary	String	Good movie
Text	String	This is a boring movie, and if your kids want it, just get it on a DVD.

6.3 Performance Analysis

6.3.1 Data Storage Space

The Data is increasing exponentially, but data storage space does not grow as fast. Therefore, to store the increasing volume of data with the limited disk space, we need to use a good data storage format which can organize the data efficiently. We have used RCFile and ORCFile for data storage format. The downloaded ~6 GB dataset from Amazon is all in plain text, and first, we loaded it into HDFS directory. Then, we used data placement structures to load data into the HIVE. Figure 15 shows the storage sizes with compression. We have used LZ4 compression technique to compress the data using RCFile and ORCFile. We can see both file format has reduced the data size significantly. RCFile reduces the data size from ~6 GB to 3.29 GB, where ORCFile reduces even more than RCFileto 2.01 GB because ORCFile uses larger data blocks than RCFile. Therefore, each block can arrange more data in column format which allows compressing each column independently.

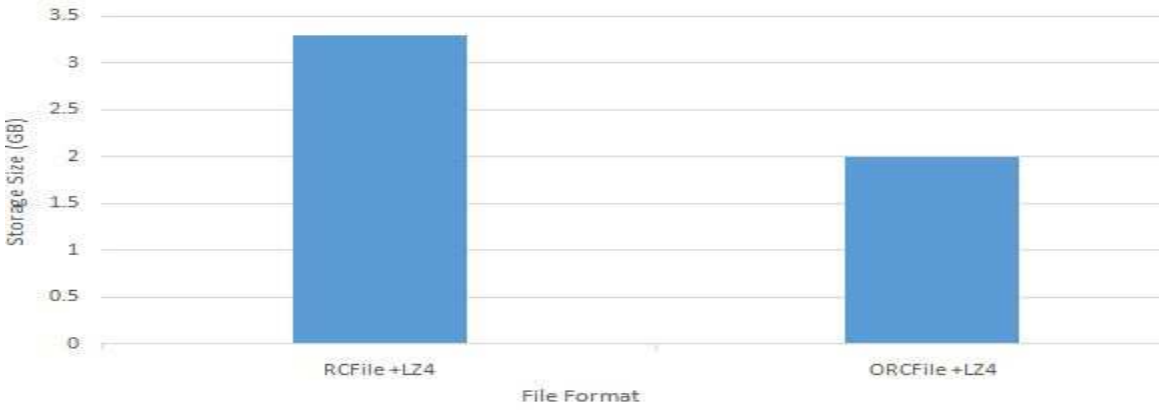


Figure 15. Storage space with compression

Figure 16 shows the storage space without compression. Here, we have arranged the dataset in Hive directory without using any general purpose compression technique. In this figure, there is not much difference between the Text file and RCFile, but ORCFile can decrease the file size significantly compare with other file formats because ORCFile uses a default compression technique (ZLIB). So, the ORCFile provides better storage efficiency than RCFile, whether using compression technique or not.

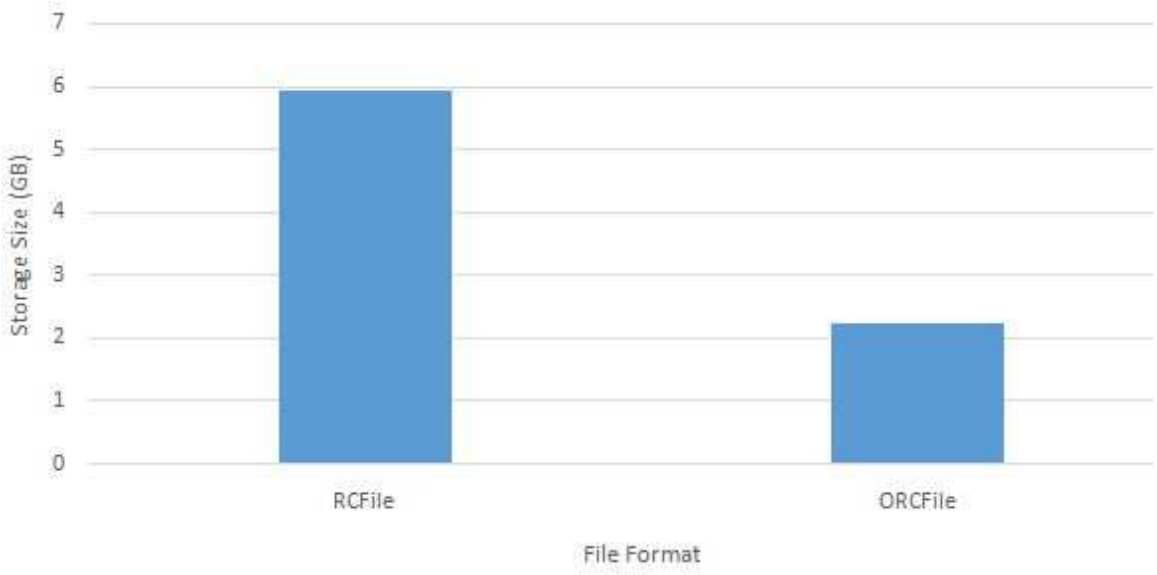


Figure 16. Storage space without compression

6.3.2 Data Loading Time

Data loading time refers to the time to load the raw data into the data warehouse system. For a data placement structure, data loading is an important factor because data is generated from multiple sources in real time with great speed. To demonstrate the data loading time of RCFile and ORCFile, we have conducted experiments with same ~6 GB dataset. In this case, we have measured both MapReduce time and the total time to finish

the job. Figure 17 shows the data loading time after compression, where we can see ORCFile takes more time to load the data than RCFile. We have taken another result shown in figure 18, which describes the loading time before using compression technique. We can see in both cases that ORCFiles needed longer loading time than RCFile. The reason is that ORCFiles have larger block size called stripe, in which each stripe contains columns that hold the raw data. So, re-organized data in each column in the stripe takes longer time than RCFile.

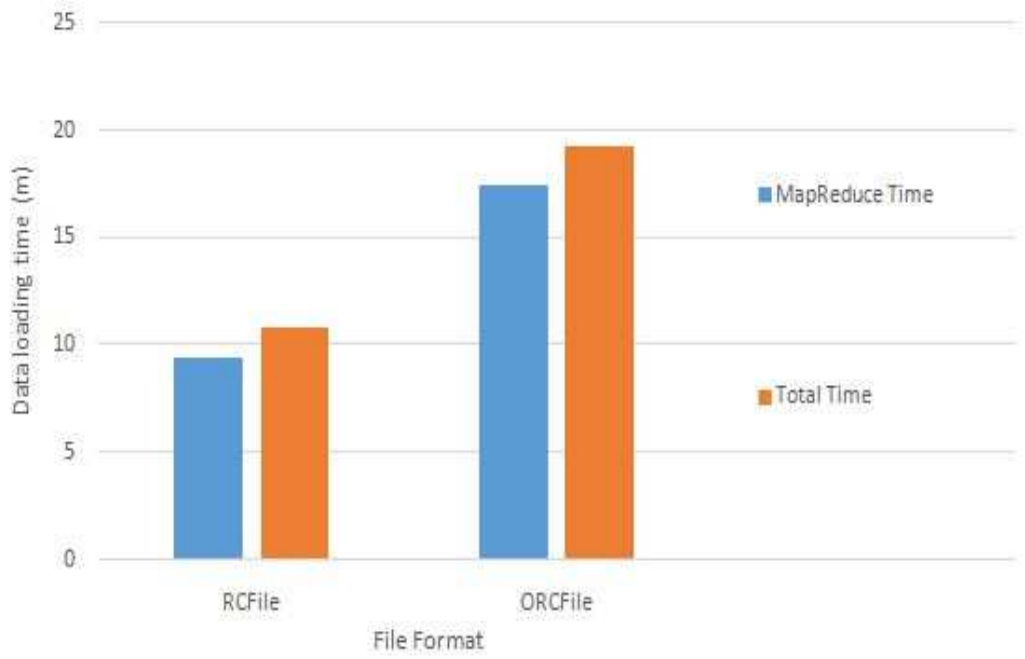


Figure 17. Data loading time with compression

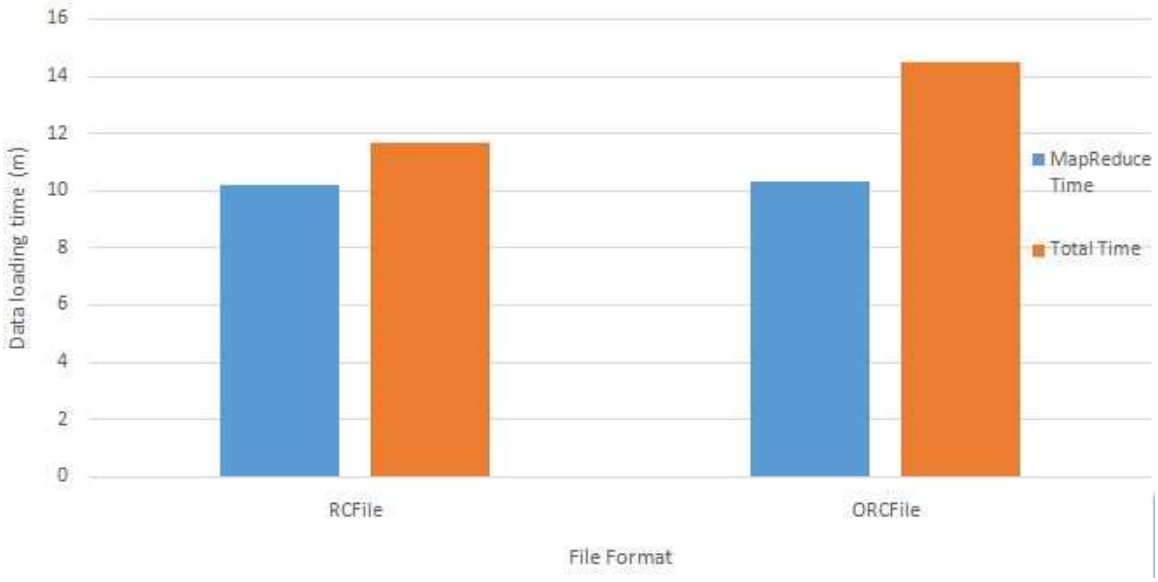


Figure 18. Data loading time without compression

6.3.3 Query Execution Time

In Big Data analytics the query execution time plays an important role to determine an efficient data storage format. In this experiments, we have executed four queries on the movie reviews' table. The queries are as follow:

- Query 1: select productid from movie_rc;
- Query 2: select reviewsummary from movie_rc where PROFILENAME = "review/profileName: Jessica Lux" and userid = "review/userId: A2EBLL2OYEQJN9";
- Query 3: select productid, userid, profilename from movie_rc where SCORE = "review/score: 50";
- Query 4: select t1.productid, t2.userid from movie_rc t1 right outer join movie_rc1 t2 on (t1.productid = t2.productid) where t1.profilename = "review/profileName: Jessica Lux";

Figure 19 shows the query execution time after data is compressed and Figure 20 shows the query execution time where we do not have used data compression technique. In both cases, ORCFile outperforms the RCFile significantly because the lazy decompression technique, which helps the ORCFile to skip a larger block of data if it does not match a query.

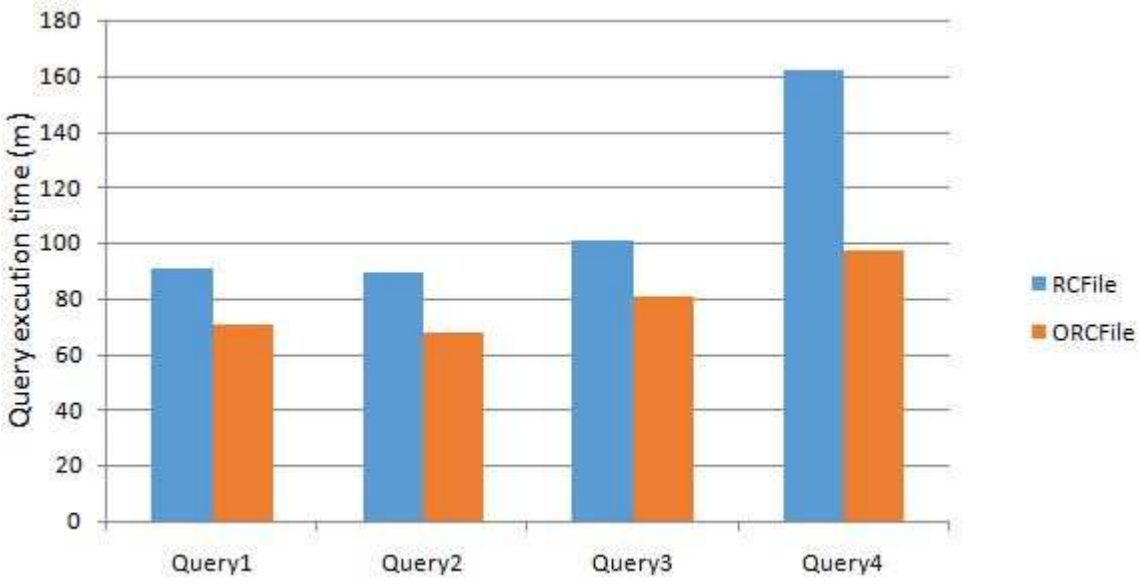


Figure 19. Query execution time with compression

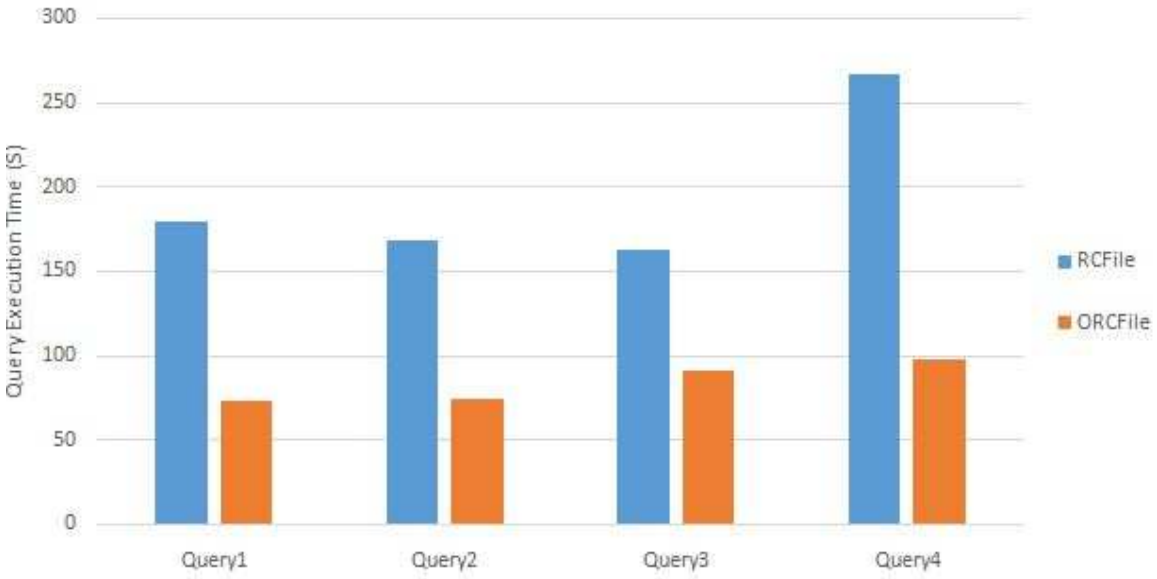


Figure 20. Query execution time without compression

6.4 RCFile and ORCFile with Different Row Group Sizes

In all of the above experiments, we have used the default configurations for both RCFile and ORCFile format. However, input/output performances are one of the major concerns to get the better result from these data storage format. Therefore, both RCFile and ORCFile allows the user to set flexible data block sizes because large data block can have better compression efficiency than a small one, where small data block may have better read or query performance than a large one. Furthermore, a large data block consumes more memory and can affect MapReduce tasks. In this experiment, we have used the same movie review database as above, and size of the dataset is 1.2 GB.

6.4.1 Data Storage Space

In this section, we have used different row group sizes for RCFile and different stripe sizes for ORCFile to demonstrate how they affect the storage space. Figure 21

shows the data storage efficiency of RCFile of different row group sizes (from 512 KB to 48 MB). Figure 22 demonstrate the data storage efficiency of ORCFile of different stripe sizes (from 4 MB to 256 MB).

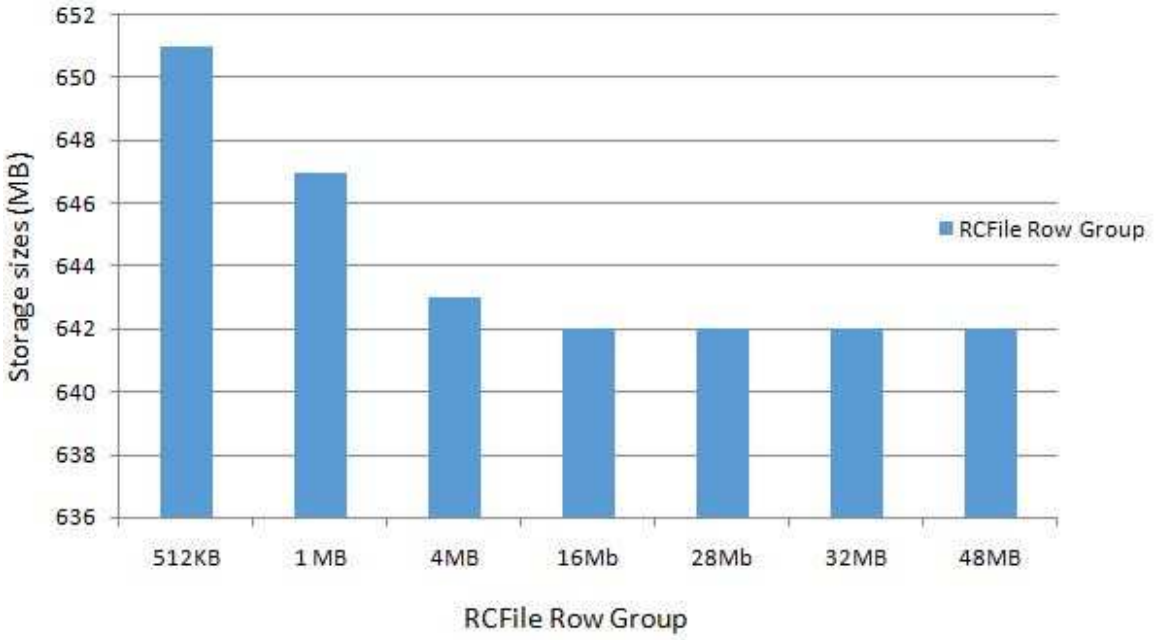


Figure 21. RCFile storage space with different row group sizes.

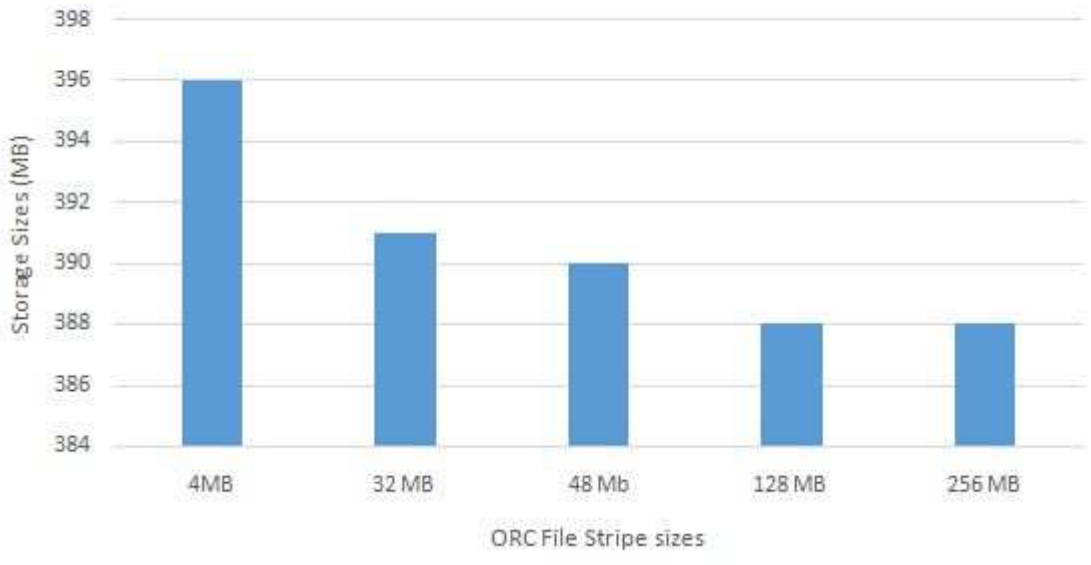


Figure 22. ORCFile storage space with different stripe sizes.

We can see that large block sizes have better compression efficiency and reduce the storage space for both file formats. However, in the RCFile, when the row group size is larger than 4 MB, the storage space almost constant. In the case of ORCFile shown in figure 8.8, it gives better compression efficiency, when the stripe sizes are larger than 48 MB. So, the ORCFiles have better storage space with larger data block than RCFile.

6.4.2 Query Execution Time

Query efficiency is an important factor of Big Data analytics for making quick decisions. In this experiment, we have evaluated the performance of lazy decompression technique of both RCFile and ORCFile. Figure 23 and 24 shows the query execution time

of RCFile and ORCFile. We have designed a query with different characteristics according to the "where" condition of a query. The query is as follow:

- Query: select reviewsummary from movie_rc where PROFILENAME = "review/profileName: Jessica Lux";

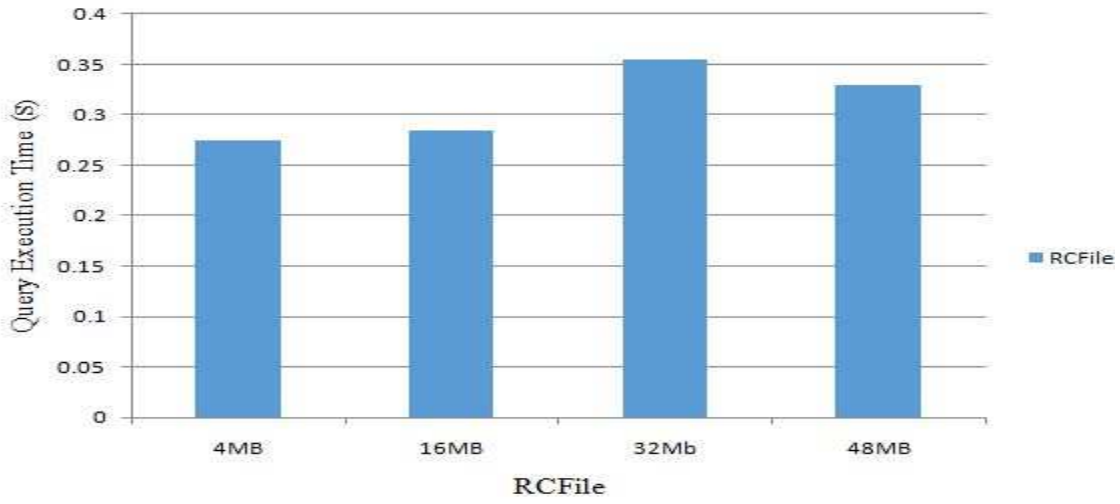


Figure 23. Query execution times of different data block sizes of RCFile.

Figure 24. Query execution times of different data block sizes of ORCFile.

From the above figure, we can see that, when row group sizes are large, RCFile gives the lower performance of query execution. The ORCFile has the better query efficiency when data block size is large (256 MB) because the ORCFile can skip large data block if it does not match the query based on lazy decompression technique.

Chapter 7

Conclusions and Future Work

The goals of Big Data analytics in large scale distributed systems are to reduce the data loading time and storage space as well as enhancing the query performance. Hence, an efficient data placement structure is an essential factor for organizing data in a proper way in MapReduce-based data warehousing to meet all the goals of Big Data analytics as mentioned above. In this research, we have presented two data placement structures, such as RCFile and ORCFile in Hive and have conducted experiments to compare these two file formats in three aspects, which are data loading time, storage space, and query execution time. Our experimental findings showed that both file formats have significant advantages than an original text file and satisfy the requirements in all three aspects Big Data analytics. Our experimental results showed that the RCFile have a major inherent advantage in data loading time over ORCFiles. Since the RCFile has small row-group size than the ORCFiles, which effectively reduces the data loading time. However, in the case of storage space and query execution time, the ORCFiles outperform the RCFile. Though both data placement structures (i.e. RCFile and ORCFile) use column-wise compression, the large row-group size inside each stripe in ORCFile can hold and compress more data at a single time and reduces more storage space for ORCFile than RCFile. We have also observed that ORCFile can skip large numbers of unnecessary columns during a query, which significantly improves query performance. Thus, the ORCFiles contain most of the performance benefits features of Big Data analytics, and we believe that the ORCFile will be the default storage choice for Big Data analytics soon.

Having presented the conclusions of best file format, we can give our attention to what else will be done in the future. The following areas deserve further study and should be pursued:

In this research work, we have used RCFile and ORCFile as data placement structure, but there are also others file format such as Avro, Parquet, which can be useful data placement structure for Big Data processing on Hadoop. Also, instead Hive, we can use others data processing tools on Hadoop such as Pig, Impala.

References

"Apache Software Framework" 2016. "Hadoop Native Libraries Guide." Accessed March 10, 2016. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/NativeLibraries.html.

Asay, Matt. 2014. "Why the World's Largest Hadoop Installation May Soon Become the Norm." Accessed March 9,2016. http://www.techrepublic.com/article/why-the-worlds-largest-Hadoop-installation-may-soon-become-the-norm/.

Bain.com 2013. "Big Data: The Organizational Challenge." Accessed March 18, 2016. http://www.bain.com/publications/articles/big_data_the_organizational_challenge.aspx.

Bertino, Elisa, Philip Bernstein, Divyakant Agrawal, Susan Davidson, Umeshwas Dayal, Michael Franklin, Johannes Gehrke, et al. 2011. "Challenges and Opportunities with Big Data." Cyber Center. Accessed March 8, 2016. http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1000&context=cctech.

Borthakur, D. 2008. "HDFS Architecture Guide." Hadoop Apache Project 1–13. Accessed March 10, 2016. http://archive.cloudera.com/cdh/3/hadoop-0.20.2-cdh3u6/hdfs_design.pdf.

Borthakur, Dhruba. 2010. "Facebook Has the World's Largest Hadoop Cluster!" Accessed March 17, 2016. http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html.

Brough, Graham. 2013. "Big Data, Big Value Huge Opportunity." Accessed March 20, 2016. http://www.sas.com/en_us/insights/articles/big-data/big-data-big-value-huge-opportunity.html.

Corrigan, David. 2012. "Big Data: Achieving Competitive Advantage through Analytics." Accessed March 15,2016. https://www-950.ibm.com/events/wwe/grp/grp037.nsf/vLookupPDFs/Calgary_Keynote_%20David_%20Corrigan%20-%20v1/$file/Calgary_Keynote_%20David_%20Corrigan%20-%20v1.pdf

Datametica. 2014. "RC/ORC File Format." Accessed March 10,2016. http://datametica.com/rcorc-file-format/.

Davenport, Tom. 2014. “Three Big Benefits of Big Data Analytics.” Accessed March 15, 2016. http://www.sas.com/en_us/news/sascom/2014q3/Big-data-davenport.html.

Dezyre. 2015. “Big-Data-and-Hadoop-Training-Hadoop-Components-and-Architecture.” Accessed March 17, 2016. https://www.dezyre.com/article/big-data-and-hadoop-training-hadoop-components-and-architecture/114.

Dobbie, Will, and Roland G. Fryer, Jr. 2013. “Getting Beneath the Veil of Effective Schools: Evidence from New York City.” American Economic Journal: Applied Economics 5 (4): 28–60. doi:10.3386/w17632.

Gunelius, Susan. 2014. “The Data Explosion in 2014 Minute by Minute – Infographic.” Accessed January 20, 2015. http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/.

Harris, Jim. 2015. “Timeliness Is the Most Important Data Quality Dimension.” Accessed March 13, 2016. http://www.ocdqblog.com/home/timeliness-is-the-most-important-data-quality-dimension.html.

“HDFS Architecture.” 2015. Accessed March 18,2016 https: //hadoop.apache.org/doc s/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

He, Yongqiang, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. 2011. “RCFile: A Fast and Space-Efficient Data Placement Structure in MapReduce-Based Warehouse Systems.” Proceedings - International Conference on Data Engineering, 1199–1208. doi:10.1109/ICDE.2011.5767933.

Hoovers. 2013. “Big Data and Analytics 101: Secrets to Data Interpretation.” Accessed March 19, 2016. http://www.hoovers.com/lc/company-information/beginners-guide-to-data-interpretation.html.

Huai, Yin, Siyuan Ma, Rubao Lee, Owen O’Malley, and Xiaodong Zhang. 2013. “Understanding Insights into the Basic Structure and Essential Issues of Table Placement Methods in Clusters.” Proceedings of the VLDB Endowment 6 (14): 1750–61. doi:10.14778/2556549.2556559.

“Internet Live User.” 2014. Accessed January 15, 2016. http://www.internetlivestats.com/internet-users/.

Issenberg, Sasha. 2012. “How President Obama’s Campaign Used Big Data to Rally Individual Voters.” Accessed March 10, 2016. http://www.technologyreview.com/featuredstory/509026/how-obamas-team-used-big-data-to-rally-voters/.

Jain, Abhishek. 2012. “Hadoop - Namenode, DataNode, Job Tracker and TaskTracker.” Accessed March 10, 2016. http://bigdata.devcodenote.com/2012/11/hadoop-namenode-datanode-job-tracker.html.

JoshBaer. 2015. “PoweredBy - Hadoop Wiki.” Accessed March 14,2016. https://wiki.apache.org/hadoop/PoweredBy.

Khan, Nawsher, Ibrar Yaqoob, Ibrahim Abaker, Targio Hashem, Zakira Inayat, Waleed Kamaleldin, Mahmoud Ali, Muhammad Alam, Muhammad Shiraz, and Abdullah Gani. 2014. “Big Data: Survey,Technologies , Opportunities , and Challenges.” The Scienctific World Journal 2014: 1–18. doi:10.1155/2014/712826.

Kobielus, James. 2014. “Next Best Expert: Collaboration of People and Machines on Big Data and Analytics.” Accessed March 16,2016. http://www.ibmbigdatahub.com/blog/next-best-expert-collaboration-people-and-machines-big-data-and-analytics.

Leverenz, Lefty. 2015. “LanguageManual ORC.” Accessed March 12,2016. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC#LanguageManualORC-orc-spec.

Murty, Arun. 2012. “Apache Hadoop YARN – Concepts and Applications.” Accessed March 10,2016. http://hortonworks.com/blog/apache-Hadoop-yarn-concepts-and-applications/.

NTTDATA. 2015. “Big Data Solutions NTT DATA.” Accessed February 21, 2016. http://www.nttdata.com/global/en/services/bds/index.html.

Oracle Corporation. 2015. Accessed March 20, 2016. “Oracle Enterprise Architecture White Paper. An Enterprise Architect’s Guide to Big Data.”

Prokopp, Christian. 2014. “ORC: An Intelligent Big Data File Format for Hadoop and Hive .” Accessed March 7,2016. http://www.semantikoz.com/blog/orc-intelligent-big-data-file-format-hadoop-hive/.

Rockwell, Drew. 2015. “ The Big Data Landscape Requires Community, Collaboration -" Accessed March 5,2016. http://data-informed.com/the-big-data-landscape-requires-community-collaboration/.

Rutledge, Pamela. 2013. “How Obama Won the Social Media Battle in the 2012 Presidential Campaign.” Accessed March 10,2016. http://mprcenter.org/blog/2013/01/how-obama-won-the-social-media-battle-in-the-2012-presidential-campaign/.

SAS. 2015. “What Is Big Data SAS.” Accessed March 21, 2016. http://www.sas.com/en_us/insights/big-data/what-is-big-data.html.

Thusoo, Ashish, Js Sarma, and Namit Jain. 2010. “Hive-a Petabyte Scale Data Warehouse Using Hadoop.” Data Engineering, 996–1005. doi:10.1109/ICDE.2010.5447738.

Tutorialspoint. 2015a. “Hadoop - Introduction.” Accessed March 2,2016. http://www.tutorialspoint.com/hadoop/hadoop_introduction.htm.

Tutorialspoint. 2015b. “Hadoop - MapReduce.” Accessed March 2,2016. http://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm.

Tutorialspoint. 2015c. “Hive Introduction.” Accessed March 2,2016. http://www.tutorialspoint.com/hive/hive_introduction.htm.

Vagata, Pamela, and Kevin Wilfong. 2014. “Scaling the Facebook Data Warehouse to 300 PB.” Accessed March 2,2016. https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/?attachment_canonical_url=https%3A%2F%2Fcode.facebook.com%2Fposts%2F229861827208629%2Fscaling-the-facebook-data-warehouse-to-300-pb%2F.

Wall, Matthew. 2014. “Big Data: Are You Ready for Blast-Off?” Accessed March 22,2016. http://www.bbc.co.uk/news/business-26383058. Accessed March 10,2016

White, Tom. 2015. Hadoop: The Definitive Guide. Vol. 54. doi:citeulike-article-id:4882841.

Wikipedia. 2016a. “Apache Hadoop.” Accessed January 11,2016. https://en.wikipedia.org/wiki/Apache_Hadoop.

Wikipedia. 2016b. “Web 2.0.” Accessed January 18,2016. https://en.wikipedia.org/wiki/Web_2.0.

Won, Jimmy. 2016. “Which Big Data Company Has the World’s Biggest Hadoop Cluster?” Accessed January 10. http://www.hadoopwizard.com/which-big-data-company-has-the-worlds-biggest-Hadoop-cluster/.

Zicari, Roberto V. 2011. “On Big Data: Interview with Shilpa Lawande, VP of Engineering at Vertica.” Accessed March 12,2016. http://www.odbms.org/blog/2011/11/on-big-data-interview-with-shilpa-lawande-vp-of-engineering-at-vertica/.

Appendices

Appendix A

Hadoop Single Node Configuration

Step 1: Update System

Following command will update Ubunto system with the latest set of packages from all the repositories.

- sudo apt-get update

Step 2: Installing java

Following command will install the latest version of java JDK.

- sudo apt-get install default-jdk

To check the java version type the following command

- java -version

Step 3: Install the SSH and generate authorized key for password less login

SSH is required to connect with Hadoop nodes and following command will install SSH.

- sudo apt-get install ssh
- ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
- cat ~/.ssh/id_dsa.pub >>~/.ssh/authorized_keys

Step 4: Disable IPV6

To check if IPV6 address is enable or disable run the following command

- cat /proc/sys/net/ipv6/conf/all/disable_ipv6

If the output is 0 means enable or if the output is 1 means disable. To disable run the following command and paste the command.

- sudo gedit /etc/sysctl.conf

net.ipv6.conf.all.disable_ipv6=1

net.ipv6.conf.default.disable_ipv6=1

net.ipv6.conf.lo.disable_ipv6=1

Step 5: Download and Untar Hadoop-2.7.1

We need to download Hadoop-2.7.1 and save to the disk.

- cd Downloads

- sudo tar -xvf hadoop-2.7.1.tar.gz

Step 6: Create a directory and move hadoop to that folder and update the java path variable

- sudo mv hadoop-2.7.1 /usr/local/hadoop
- update-alternatives --config java (Java path is /usr/lib/jvm/java-7-openjdk-amd64)
- sudo gedit hadoop-env.sh

Step 6: Configure the Hadoop variable

Run the follwing command and type the Hadoop variables to the end of the bash file

- sudo gedit ~/.bashrc

hadoop variables

export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

export HADOOP_HOME=/usr/local/hadoop

export PATH=$PATH:$HADOOP_HOME/bin

exportPATH=$PATH:$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

export HADOOP_OPTS="$HADOOP_OPTS-Djava.library.path=$HADOOP_HOME/lib/native"

- source ./~bashrc (update the bashrc file)

Step 7: Create name node and data node directory

- sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
- sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
- sudo chown hasan:hasan -R /usr/local/hadoop/

Step 8: Configure the Hadoop directories where the data file stored.

Type the following command to go the right directory.

- cd /usr/local/hadoop/etc/hadoop

- sudo gedit mapred-site.xml.template (open file and type the following lines between configuration tag)

sudo gedit mapred-site.xml

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

- sudo gedit yarn-site.xml(open file and type the following lines between configuration tag)

<property>

<name>yarn.nodemanager.auxservices</name>

<value>mapreduce_shuffle</value>

<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.shuffleHandler</value>

</property>

- sudo gedit core-site.xml(open file and type the following lines between configuration tag)

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

- sudo gedit hdfs-site.xml(open file and type the following lines between configuration tag)

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/usr/local/hadoop/hadoop_data/hdfs/namenode</value>

</property>

<property>

<name>dfs.datanode.name.dir</name>

<value>file:/usr/local/hadoop/hadoop_data/hdfs/datanode</value>

</property>

Step 9: Format name node and start the single node hadoop Hadoop cluster

- hdfs namenode -format
- start-all.sh
- hadoop version
- jps

If everything is successful, then the following services will be appeared:

DataNode

ResourceManager

Jps

NodeManager

NameNode

SecondaryNameNode

At this point Hadoop single node cluster is installed.

Appendix B

Hadoop Multi-node Configuration

Step 1: Follow the appendix a procedure to set up single node Hadoop in required numbers of machine and select a machine for master (name node) and others machines for slaves (datanode). From master machine, the following command will open a file and write all the hosts network address and host name.

- sudo gedit /etc/hosts

192.168.0.1 hadoopmaster
192.168.0.2 hadoopslave1
192.168.0.3 hadooplslave2
192.168.0.4 hadooplslave3
192.168.0.5 hadooplslave4

Step 2: Write the hostname for master machine by typing the following command

- sudo gedit /etc/hostname (open file and put master node name)

master

Step 3: Configure the Hadoop directories for multinode cluster. Do the following changes for all machines (master and slaves).

- sudo gedit mapred-site.xml.template (open file and type the following lines between configuration tag)

  <property>
  <name>mapred.job.tracker</name>
  <value>hadoopmaster:54311</value>
  </property>
  <property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
  </property>

- sudo gedit yarn-site.xml(open file and type the following lines between configuration tag)

  <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>hadoopmaster:8025</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value> hadoopmaster master:8030</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value> hadoopmaster master:8050</value>
</property>

- sudo gedit core-site.xml(open file and type the following lines between configuration tag)

  <property>
  <name>fs.defaultFS</name>
  <value>hdfs://hadoopmaster:9000</value>
  <description>NameNode URI</description>
  </property>

- sudo gedit hdfs-site.xml(open file and type the following lines between configuration tag)

  <property>
  <name>dfs.replication</name>
  <value>2</value>
  </property>
  <property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/datanode</value>
  <description>DataNode directory</description>
  </property>
  <property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/namenode</value>
  <description>NameNode directory for namespace and transaction logs storage.</description>

```
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
<property>
<name>dfs.datanode.use.datanode.hostname</name>
<value>false</value>
</property>
<property>
<name>dfs.namenode.datanode.registration.ip-hostname-check</name>
<value>false</value>
</property>
<property>
        <name>dfs.namenode.http-address</name>
<value>hadoopmaster:50070</value>
<description>Your NameNode hostname for http access.</description>
</property>
```

Step 4: From master machine write down the master and slaves machine name or IP addresses.

- sudo gedit /usr/local/hadoop/etc/hadoop/masters

master or IP address

- sudo gedit /usr/local/hadoop/etc/hadoop/slaves

master
slave1
slave2
slave3

Step 5: Remove the data node from master machine because master machine work as a name node. The following command will open the HDFS file and just delete the portion of data node.

- sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml

Step 6: From slave machine, give the host name in every slave machine and restart to make the changes.

- sudogedit /etc/hostname

hadoopslave1

Step 7: Remove the name node from master machine because slave machines work as a data node. The following command will open the HDFS file and just delete the portion of name node.

- sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml

Step 8: Delete the Hadoop data directories, what we created in the single node cluster. In master node, we need to create only name node directory and for slave nodes is data node directories.

- master machine
- sudo rm -rf /usr/local/hadoop/hadoop_data/
- sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
- sudo chown -R hasan:hasan /usr/local/hadoop

- login slave 1 (do all other slave): (do all other slave)
- sudo rm -rf /usr/local/hadoop/hadoop_data/
- sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
- sudo chown -R hasan:hasan /usr/local/hadoop

Step 8: The following commands will make sure that master machine can login in password less with slave machines.

ssh-copy-id -i ~/.ssh/id_dsa.pub hasan@192.168.0.1
ssh-copy-id -i ~/.ssh/id_dsa.pub hasan@192.168.0.2
ssh-copy-id -i ~/.ssh/id_dsa.pub hasan@192.168.0.3
ssh-copy-id -i ~/.ssh/id_dsa.pub hasan@192.168.0.4

Step 9: Format name node and start the multi node hadoop Hadoop cluster

- hdfs namenode -format
- start-all.sh
- hadoop version
- jps

If everything is successful, then the following services will be appeared:

DataNode

ResourceManager

Jps

NodeManager

NameNode

SecondaryNameNode

At this point Hadoop multi node cluster is installed.

Appendix C

Hive Configuration

Step 1: Download the latest version of Hive and the following commands will untar and move to the Hive folder

- sudo tar -xvf apache-hive-1.1.0-bin.tar.gz
- sudo mv apache-hive-1.1.0-bin /usr/local/hive

Step 2: Configure the Hive directory.

- sudogedit ~/.bashrc (open file and type the following lines at the end of file)

```
export HIVE_HOME=/usr/local/hive
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HIVE_CLASS_PATH=$HIVE_CONF_DIR
export PATH=$HIVE_HOME/bin:$PATH
export HADOOP_USER_CLASSPATH_FIRST=true
```

- Update the bashrc file by following command

```
source ~/.bashrc
```

Step 3: Copy the Hive jar file Hive library to Hadoop library.

- cp hive-1.2.0/lib/jline-2.12.jar $HADOOP_HOME/share/hadoop/yarn/lib/

Step 4: To start Hive first start Hadoop and the type the following command to start Hive

- hive

At this point Hive is installed in master machine

Appendix D

HiveQL and Table Schema

Table Creation:

```
CREATE TABLE MOVIE_TEXT (PRODUCTID STRING,USERID
STRING,PROFILENAME STRING,HELPFULNESS STRING,SCORE STRING,
REVIEWTIME STRING,REVIEWSUMMARY STRING,REVIEWTEXT
STRING)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Load data into table:

```
load data local inpath '/home/hasan/Desktop/movie.txt' into table movie_text;
```

Create RC Table:

```
CREATE TABLE MOVIE_RC(PRODUCTID STRING,USERID
STRING,PROFILENAME STRING,HELPFULNESS STRING,SCORE STRING,
REVIEWTIME STRING,REVIEWSUMMARY STRING,REVIEWTEXT
STRING)STORED AS RCFILE;
```

Set compression and row group sizes:

```
setmapred.output.compress=true;
sethive.exec.compress.output=true;
SET hive.io.rcfile.record.buffer.size =512000;
set mapred.output.compression.codec=org.apache.hadoop.io.compress.Lz4Codec;
setio.compression.codecs=org.apache.hadoop.io.compress.Lz4Codec
```

Load data into RC table:

```
INSERT OVERWRITE table movie_rc select * from movie_text;
```

Create ORC Table:

```
CREATE TABLE MOVIE_ORC(PRODUCTID STRING,USERID
STRING,PROFILENAME STRING,HELPFULNESS STRING,SCORE STRING,
REVIEWTIME STRING,REVIEWSUMMARY STRING,REVIEWTEXT STRING)
STORED AS ORC TBLPROPERTIES "orc.stripe.size"="64000000");
```

Set compression and row group sizes:

```
set hive.exec.orc.compression.strategy= COMPRESSION;
set mapred.output.compress=true;
set hive.exec.compress.output=true;
set mapred.output.compression.codec=org.apache.hadoop.io.compress.Lz4Codec;
```

Load data into ORC table:

```
INSERT OVERWRITE table movie_orc select * from movie_text;
```