UNIVERSITY OF VAASA

SCHOOL OF TECHNOLOGY AND INNOVATIONS

WIRELESS INDUSTRIAL AUTOMATION

Adeyemi Adesokan

PERFORMANCE ANALYSIS OF HADOOP MAPREDUCE AND APACHE SPARK FOR BIG DATA ANALYSIS

Master's thesis for the degree of Master of Science in Technology that has been submitted for inspection, Vaasa 18 May, 2020.

Supervisor Professor Mohammed Elmusrati

Instructor Professor Timo Mantere

FOREWORD

I would like to appreciate my mother, Mrs Olukemi Adesokan, who has always supported me. I want to acknowledge my supervisor Professor Muhammed Elmusrati. Also, special thanks to all the teachers that have shared their knowledge with me. I dedicate this research work to my dear late sister, Olaitan Adesokan. I love you so much.

LIST OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

APIs	Application Programming Interface
EC2	Amazon Elastic Compute Cloud
AWS	Amazon Web Services
CDH	Cloudera Distribution for Apache Hadoop
CM	Cloudera Manager
HADOOP	High-Availability Distributed Object-Oriented Platform
HDFS	Hadoop Distributed Files Systems
HPCC	High Performance Computing Cluster
IaaS	Infrastructure-as-a-Service
IOT	Internet of Things
IT	Information Technology
PaaS	Platform-as-a-Service
RDD	Resilient Distributed Datasets
SaaS	Software-as-a-Service
SSH	Secure Socket Shell
VM	Virtual Machine
VPC	Virtual Private Cloud

UNIVERSITY OF VAASA
Faculty of technology School of Technology
Author: Adesokan Adeyemi
Topic of the Thesis: Performance Analysis of Hadoop MapReduce And Apache Spark for Big Data
Supervisor: Professor Muhammed Elmusrati
Instructor: Professor Timo Mantere
Degree: Master of Science in Technology
Major of Subject: Wireless Industrial Automation
Year of Entering the University: 2015
Year of Completing the Thesis: 2020 **Pages:** 72

ABSTRACT

In the recent era, information has evolved at an exponential rate. In order to obtain new insights, this information must be carefully interpreted and analyzed. There is, therefore, a need for a system that can process data efficiently all the time. Distributed cloud computing data processing platforms are important tools for data analytics on a large scale. In this area, Apache Hadoop (High-Availability Distributed Object-Oriented Platform) MapReduce has evolved as the standard. The MapReduce job reads, processes its input data and then returns it to Hadoop Distributed Files Systems (HDFS). Although there is limitation to its programming interface, this has led to the development of modern data flow-oriented frameworks known as Apache Spark, which uses Resilient Distributed Datasets (RDDs) to execute data structures in memory. Since RDDs can be stored in the memory, algorithms can iterate very efficiently over its data many times.

Cluster computing is a major investment for any organization that chooses to perform Big Data Analysis. The MapReduce and Spark were indeed two famous open-source cluster-computing frameworks for big data analysis. Cluster computing hides the task complexity and low latency with simple user-friendly programming. It improves performance throughput, and backup uptime should the main system fail. Its features include flexibility, task scheduling, higher availability, and faster processing speed. Big Data analytics has become more computer-intensive as data management becomes a big issue for scientific computation. High-Performance Computing is undoubtedly of great importance for big data processing. The main application of this research work is towards the realization of High-Performance Computing (HPC) for Big Data Analysis.

This thesis work investigates the processing capability and efficiency of Hadoop MapReduce and Apache Spark using Cloudera Manager (CM). The Cloudera Manager provides end-to-end cluster management for Cloudera Distribution for Apache Hadoop (CDH). The implementation was carried out with Amazon Web Services (AWS). Amazon Web Service is used to configure window Virtual Machine (VM). Four Linux Instances of free tier eligible t2.micro were launched using Amazon Elastic Compute Cloud (EC2). The Linux Instances were configured into four cluster nodes using Secure Socket Shell (SSH).

A Big Data application is generated and injected while both MapReduce and Spark job are run with different queries such as scan, aggregation, two way and three-way join. The time taken for each task to be completed are recorded, observed, and thoroughly analyzed. It was observed that Spark executes job faster than MapReduce.

KEYWORDS: Apache Spark, Big Data, CDH, EC2, HDFS, Hadoop, MapReduce.

1 INTRODUCTION

Big Data is growing rapidly in terms of volume and speed. Large quantities of data are generated globally, mostly unstructured in different forms. Furthermore, new technologies have evolved to extract complex data such as machine-generated information, device data and sensor data. The quality of these large data depends on its analysis. There is therefore a need for proper analysis of these unstructured data for new insights. (Verma, et al., 2016).

Big data is widely used to refer to large and complex data collection that exceeds the processing capabilities of conventional data management systems. Big Data can be described as an increased data volume that is difficult to store, process, and analyze using conventional computing tools. There are three main dimensions of Big Data known as 3Vs (Volume, Velocity, and Variety). Due to vast amounts of data in different formats and varying quality that needs to be processed quickly, more Vs have been introduced. The two additional Vs are Veracity and Valence. (Aziz, et al., 2018)

Volume refers to the vast quantities of data produced in our digitalized world every second. It is primarily all data types that are produced from different sources and expand over time continuously. Before three decades ago, only 10^6 bytes were obtainable from a floppy disk. Nowadays, storage and processing cover exabytes (10^{18}) or even zettabytes (10^{21}). The data variety refers to ever-increasing forms of data gathered by sensors, smartphones, images, and text. This format of data can be structured, semi-structured and unstructured. Data structures are critical in managing vast amounts of data. It is a specialized format for organizing, processing, extracting, and storing data. The data velocity refers to the speed at which information is produced and transmitted from one point to the next. The content of information is not constant and changes continuously due to additional data sets and data streaming from multiple sources. (Aziz, et al., 2018) Figure 1 shows a brief description of the five characteristics.

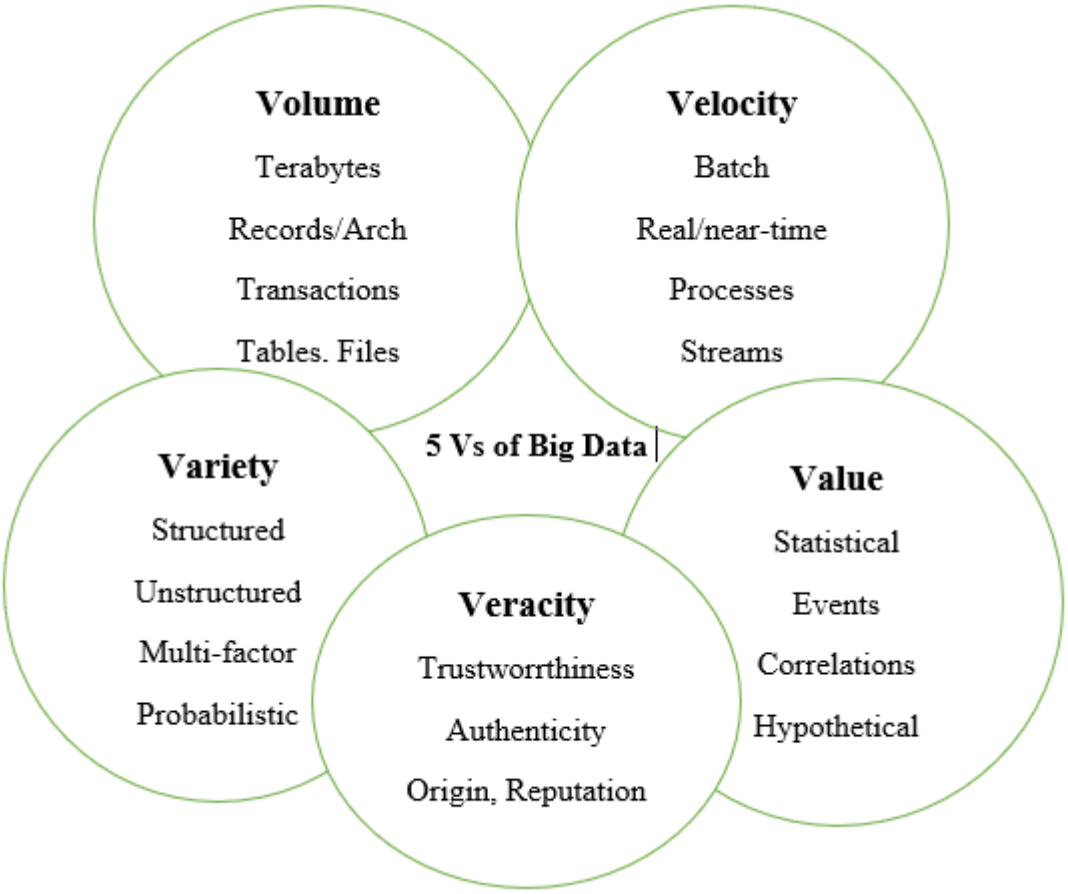

Figure 1 5Vs of Big data and their Characteristics.

Hadoop MapReduce has emerged as a highly effective Big Data analysis tool. Nevertheless, it has been speculated that MapReduce is not quick in response to real-time analysis, unlike Apache Spark. (Verma, et al., 2016).

MapReduce processes data in parallel with the framework reduction map. This led to the development of the Hadoop framework for distributed computing across several nodes. The main setback was the processing of redundant datasets maybe, takes a considerable amount of time. (Hazarika, et al., 2017).

Today, as organizations face the challenges for real-time data analysis, a new open-source Hadoop data processing tool, Apache Spark has been introduced. Apache Spark was developed in 2009 and open-sourced in 2010. (Verma, et al., 2016).

It stores data in a fault-tolerant system in Resilient Distributed Datasets (RDD). At the moment, Spark relies on Hadoop Distributed Files Systems (HDFS) and cloud-based storage for computing and does not have its own storage. It is considered a sophisticated and faster analytics tool. (Hazarika, et al., 2017).

1.1 Research Issues

Hadoop framework serves many of Big Data applications such as machine learning and crawling. There is various research work going on areas such as scheduling; mapping reduces optimization, data localization, sort optimization, and speculative execution. Hadoop MapReduce has been speculated to be inefficient for iterative computation. Apache Spark is considered a top-level large-scale sorting Big Data tools. There are lots of research work going on in various fields of Apache Spark, such as adding more machine learning algorithms and an RDD system. Spark is an in-memory processing Big Data analytic tools. It has been speculated to become inefficient once the data size exceeds the primary memory capacity.

1.2 Thesis Motivation and Objectives

Big Data success depends on its analysis. Several organizations face numerous challenges in timely and effective processing and analysis of the wealth of data. Opportunities are however available with the proper technology platform for timely storage and analysis of Big Data. The current research shows that the appropriate use of a parallel and distributed technology platform could be the solution. These platforms include two cluster-based computing frameworks that are widely used for large-scale data processing. Hadoop MapReduce and Apache Spark are the two widely used platforms.

Implementation of MapReduce Hadoop has become massive for storing, scanning, managing, and processing large volumes of data. MapReduce tackles major challenges

of distributed computations involving high scalability redundancy built-in, and safety failure. However, MapReduce is not quick in responding to real time data analysis. The Apache Spark performs sophisticated data analytics at rapid lightning speed. It focuses more on efficient processing of the distributed data. It supports iterative processing and storage in a fault-tolerant on large clusters.

The aim of this study is to compare the usability of two popular distributed computing systems: Apache Hadoop MapReduce and Apache Spark in cloud computing cluster setup. The focus is on data processing in the cloud cluster computing. The main objective is to observe and analyze the effectiveness of both the MapReduce and Spark when subjected to the same big data in a different scenario within the cloud distributed computing. The result gives more insight into how these two cluster-based analytics tools behave within cloud computing for data analysis.

1.3 Thesis Contribution

B. Akil. et al., (2017) compares three data analytic tools: Apache Hadoop; Apache Flink; and Apache Spark. The report was based on usability and execution of all the three computing platforms. They initiated three different tasks with each of these prominent data processing platforms. The participants were then asked to fill a survey data on the satisfaction levels with three analytic tools. The MapReduce was particularly found to be very difficult in debugging, while Apache Spark environment covers basic usage environment effectively. The report concluded that Apache Spark and Flink were preferable over the MapReduce.

A. Verma. et al., (2016) carried out a comparison study between Hadoop MapReduce and Apache Spark. They observed that there is inefficiency in MapReduce execution when ingested with a large scale of data. The Apache Spark was found to be very effective in this regard. The report concluded that Apache Spark's effectiveness is due to its in-memory processing capability. This also aided its ability for batch processing, streaming processing, and machine learning.

M. Khan. et al., (2017) compared the programming models of both MapReduce and Apache Spark for computational efficiency. They ingested three big data applications while varying the input of the dataset. In all the cases they experimented, Apache Spark was found to be more efficient. The report observed that Spark architecture is designed for parallel datasets processing, such as a machine learning algorithm. Its in-memory data processing enables data to be cached in the memory. The report concluded that this primitive optimizes iterative computation and reduces datasets access latency in Apache Spark.

1.4 Thesis Outline

The first part of this research work will highlight theoretically Big Data Analytic Tools, Big Data cloud, HDFS, Hadoop Ecosystem, Hadoop MapReduce, and Apache Spark. Chapter 3 of this research work will highlight the cloud computing services with a focus on AWS, EC2 and the Cloudera Manager. The performance metric will be discussed briefly. Chapter 4 focuses on implementation and benchmarking. We will discuss the results in Chapter 5. Also, the results will be fully analysed. We will conclude this thesis work with Chapter 6, and future work discussed.

2 BIG DATA ANALYTIC

Extracting useful information from vast digital data sets involves smart, robust analytics services, programming tools, and applications. The abundance of data stores, websites, sources of audio and video, tweets and blogs create an enormous amount of complex and widespread digital data. Efficient means to create, store and share this information are now in place, which also stimulates the growth of data. However, the extraction of useful knowledge from vast digital databases involves intelligent and scalable analytics systems, programming tools and applications. (Talia, 2013).

To deliver optimum performance, Big Data Analytics uses computed-intensive data collection algorithms that involve powerful high-performance processing. Infrastructure and services for cloud computing can act as an important tool for meeting the computational and data storage needs of large data analytics applications. This trend allows clouds to become an infrastructure for integrating universal and flexible systems in data analytics. Addressing and extracting value from cloud-based Big Data Calls for advanced analytics. (Talia, 2013).

There are three models associated with the implementation of Big Data Analytics services solutions in the cloud, namely:

- Data Analytics Software as a Service.
- Data Analytics Platform as a Service.
- Data Analytics Infrastructure as a Service.

The Data Analytic Software as a Service model provides end-users with full broad data analytics solutions that can exploit cloud scalability in both data storage and processing power to analyze large and complex datasets. The Data Analytics Platform as a Service model offers suites and environments for data analysis software where data mining and

scalable analytics tools can be built. As a service model , the data analytics infrastructure can be used to construct collections of virtualized hardware and software tools to run data analysis systems. (Talia, 2013).

2.1 Big Data Analytic Tools

Big Data Analytics employs several tools, namely Hadoop, High-Performance Computing Cluster (HPCC), Hurricane, HBase, and Grid Gain. These are used to enhance the different factors involved in Big Data development and computer system usability. Big data analytical tools have five main approaches for analyzing data and generating information. Each of these methodologies plays a key role in the uncovering of hidden relations. They are as follows:

I. **Discovery tools**: It is a tool that is essential for the regular, intuitive discovery and review of data from any combination of structured, unstructured, and semi-structured sources during the information life cycle. Such tools allow research to be carried out alongside conventional systems of BI origin. With the support of BI software, users can draw new data, come to a valid or useful conclusion and make informed decisions quickly. (R. K. Chawda and G. Thakur, 2016).

II. **Business Intelligence (BI) Tools**: This is very useful for evaluation, reporting and performance management, particularly for relational data type data storage.

III. **In-Database Analytics**: such analytical techniques are applied directly to the database, which require data processing. It includes a variety of techniques, such as credit scores, detection of fraud, patterns, and results relations.

IV. **Hadoop**: It is the most common open-source framework for computing, which is scalable, efficient, and distributed. It is very useful for pre-processing information or finding pieces of information for identity macro patterns. In the main, all organizations use Hadoop as an ancestor to advance analytical forms. (R. K. Chawda and G. Thakur, 2016).

V. **Decision Management**: This consists of computational modelling, self-learning and rules of the market for taking informed action based on the current context. This type of analysis allows for different recommendations across different channels, optimizing each user 's quality for interaction. (R. K. Chawda and G. Thakur, 2016).

2.2 Big Data Cloud

The study of Big Data is like exploring our planet from an entirely new perspective, the value that emerges from this study can be related to the exploration of a parallel world that has remained a mystery for humanity over the years. While Cloud Computing has been on the rise, data scientists predict that Big Data will be the next "Huge thing" in the Information Technology (IT) world. The large volume of data usage suggests new challenges and opportunities for future studies. When Big Data Techniques is being used to store and analyse cloud data, this cloud infrastructure can be considered Big Data Cloud. Figure 2 shows a basic description of the Big Data Cloud. (Khorshed, et al., 2015).

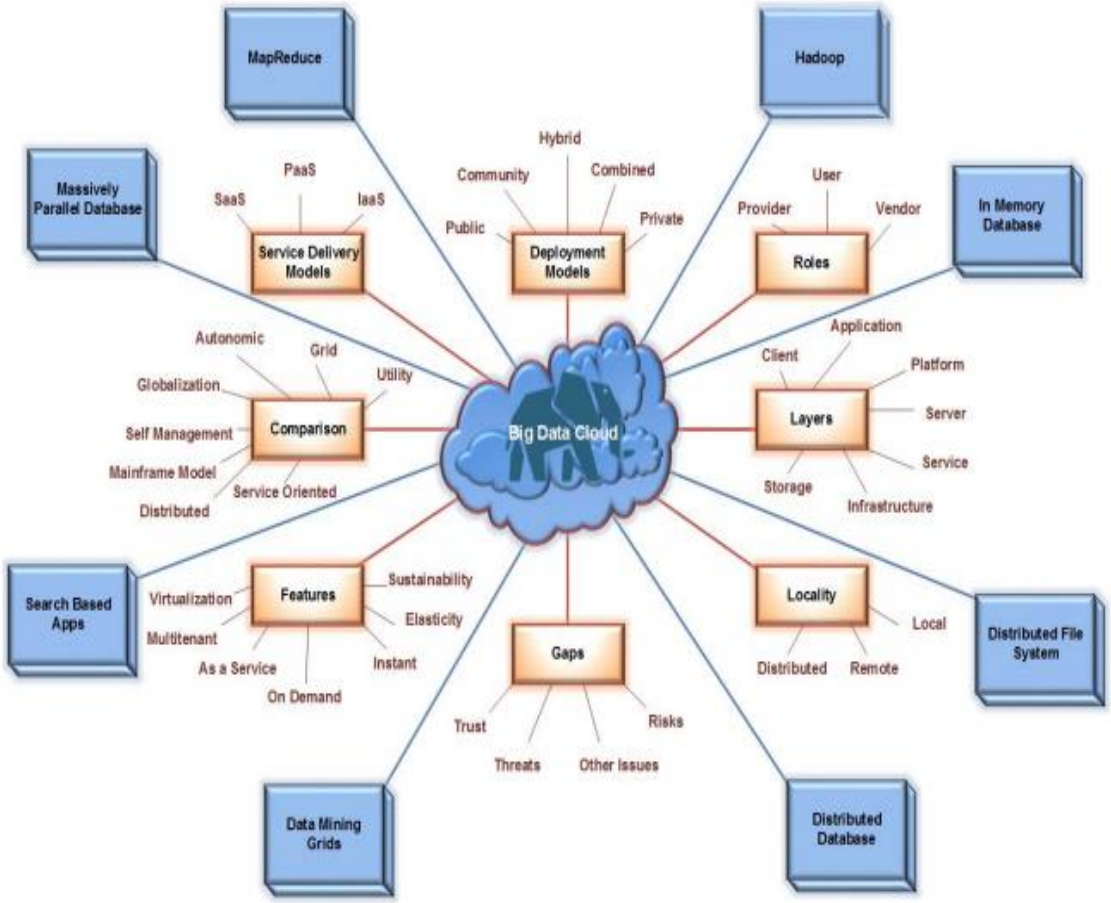

Figure 2 Description of Big Data Cloud. (Khorshed, et al., 2015).

Cloud is the term used by the Internet as a framework for the use of cloud storage infrastructure to collect and process data instead of local servers or personal computers. Cloud computing has essentially evolved as a heterogeneous cloud environment for delivering end-user storage services and is now evolving as an Internet of Things (IoT) platform. Cloud computing has been the phenomenon towards the most efficient and prominent service-oriented computing platform in the last two decades. This eventually turned cloud computing into innovative technology refinement, with the most common architecture being the PaaS system and the SaaS software. (A. K. Manekar and G. Pradeepini, 2015).

2.3 Apache Hadoop Ecosystem

In 2004, a paper was published by Google on their in-house computing system called MapReduce. The following year, Yahoo launched an open-source solution based on Hadoop architecture. In recent years, other frameworks and resources have been made available to the public as open-source projects. The Hadoop ecosystem comprises of an increasing number of open-source tools. Provide opportunities to choose the right tool for the right tasks for improved performance at lower costs. Nowadays, there are over 100 open-source big data initiatives, and this number keeps growing. A lot of organization relies on Hadoop. Figure 3 shows a complete description of a Hadoop Ecosystem. (The Hadoop Ecosystem: Welcome to the zoo!:).

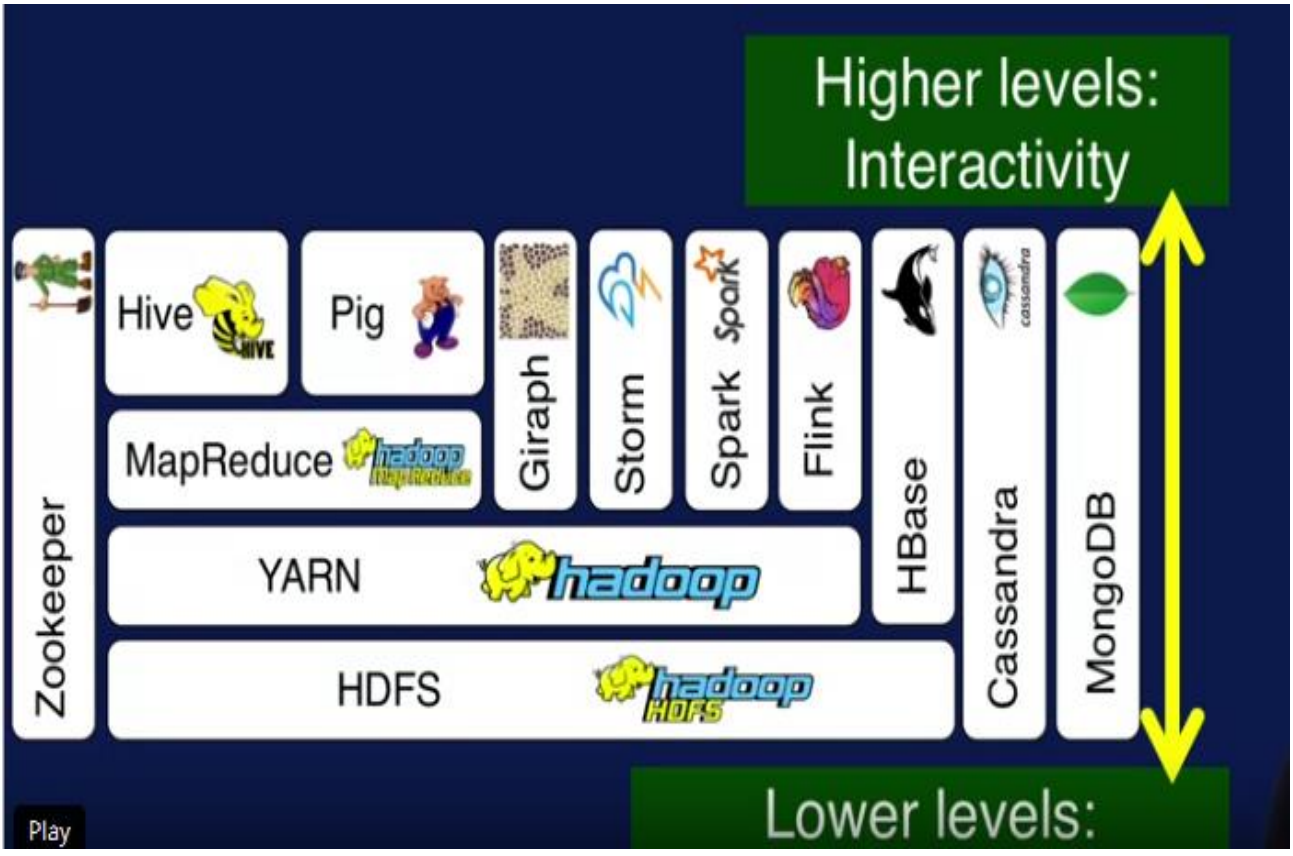

Figure 3. A complete description of Hadoop Eosystem (The Hadoop Ecosystem: Welcome to the zoo!:).

Let’s view the set of tools in the Hadoop environment as a layer diagram, with so many modules and tools available. We arrange them as a layer diagram to consider their capabilities. The schematic of the layer is arranged vertically depending on the interface. Low-level interface, processing and scheduling are at the bottom. High-level languages and the interactivity are at the top. In a layer diagram, the unit uses the functions or features of the modules in the layer below it. Typically, components on the same level do not interact or communicate. (The Hadoop Ecosystem: Welcome to the zoo!:).

2.4 Hadoop Distributed File System

This is an essential part of the Hadoop Ecosystem. HDFS is Hadoop's main storage system. HDFS is a Java-based file system that offers scalable, fault-tolerant, efficient and cost-effective data storage for Big Data. It is a distributed file system running on hardware. HDFS has already been configured with a default configuration for several installations. In most cases, it is needed for the configuration of large clusters. Hadoop communicates with HDFS directly through shell-like commands. (Hadoop Ecosystem and Their Components – A Complete Tutorial by DataFlair Team, 2019).

HDFS has two main components namely.

I. NameNode: This is also known as the Node Master. NameNode does not store any dataset or data. It stores Metadata, i.e. the number of nodes, their location, on which Rack the DataNode is located, and other information. It is made up of files and directories. The following are the tasks executed by NameNode; Manages file system namespace, Regulation of client’s access to files and Execution of file system execution.

II. DataNode: This is also known as the Slave. HDFS Datanode is liable for storing real HDFS data. It conducts a read and write operation when requested by the clients. The Datanode Replica Block consists of 2 directories on the file system. The first one is for data and the second is for storing the metadata. HDFS

Metadata provides data checksums. When initialized, each Datanode links to its corresponding Nameode and creates a handshake. The validation of the namespace identity and the application version of DataNode takes place through handshaking. It performs mainly these two tasks: The block replica creation and deletion obeying NameNode instruction. It also manages the data storage of the system. (Hadoop Ecosystem and Their Components – A Complete Tutorial by DataFlair Team, 2019).

2.5 Hadoop MapReduce

Hadoop MapReduce is the main feature of the Hadoop ecosystem providing storage of the data. MapReduce is a simple application software platform that processes a huge amount of structured and unstructured data contained in the HDFS. Its programs are designed in parallel, making them very efficient for carrying out large-scale data analysis with many cluster machines. This parallel processing improves the cluster's reliability and performance. The features of MapReduce include scalability, simplicity, and tolerance to failures. Figure 4 shows the basic working principle of Hadoop MapReduce. (Hadoop Ecosystem and Their Components – A Complete Tutorial by DataFlair Team, 2019).

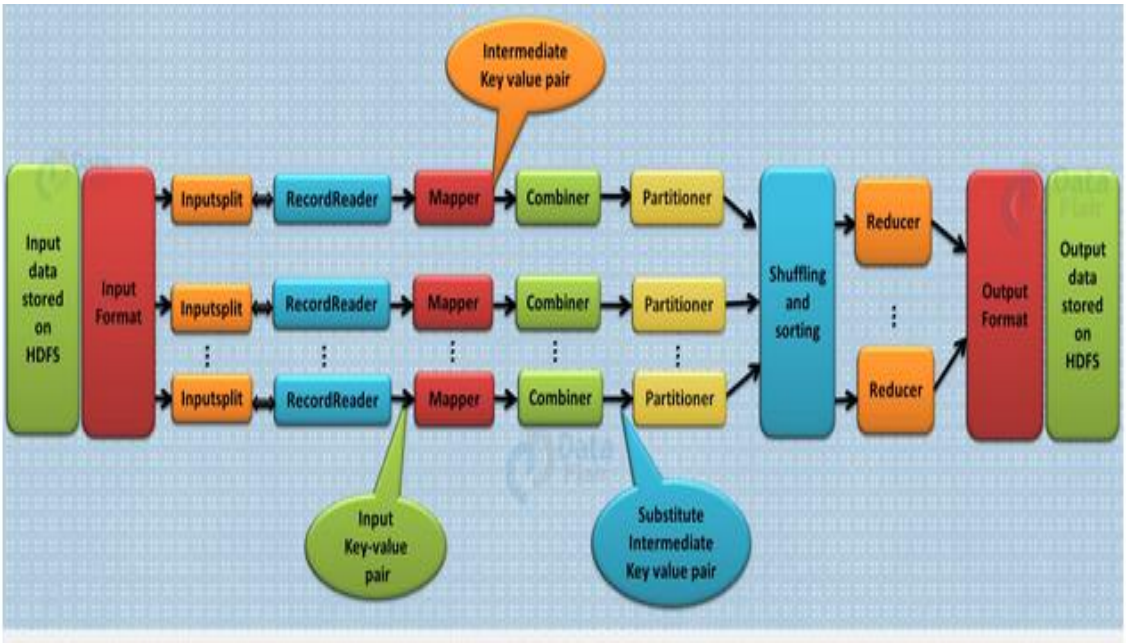

Figure 4. Working Principle of Hadoop MapReduce. (Hadoop Ecosystem and Their Components – A Complete Tutorial by DataFlair Team, 2019).

2.5.1 MapReduce Framework

Hadoop communicates with structured, unstructured, and semi-structured data. In Hadoop, once the schema is static, it works directly on the column instead of the keys and values, but if the schema is not static, it must work on the keys and values. Keys and values are not the inherent properties of the data but are chosen by the user who analyses the data. MapReduce work by breaking down the processing into two phases: Map Phase and Reduce Phase. Each phase has key value sets for inputs and outputs. Figure 5 shows the basic concepts of the key-value pair. (Learn the Concept of Key-Value Pair in Hadoop MapReduce by DataFlair Team, 2018).

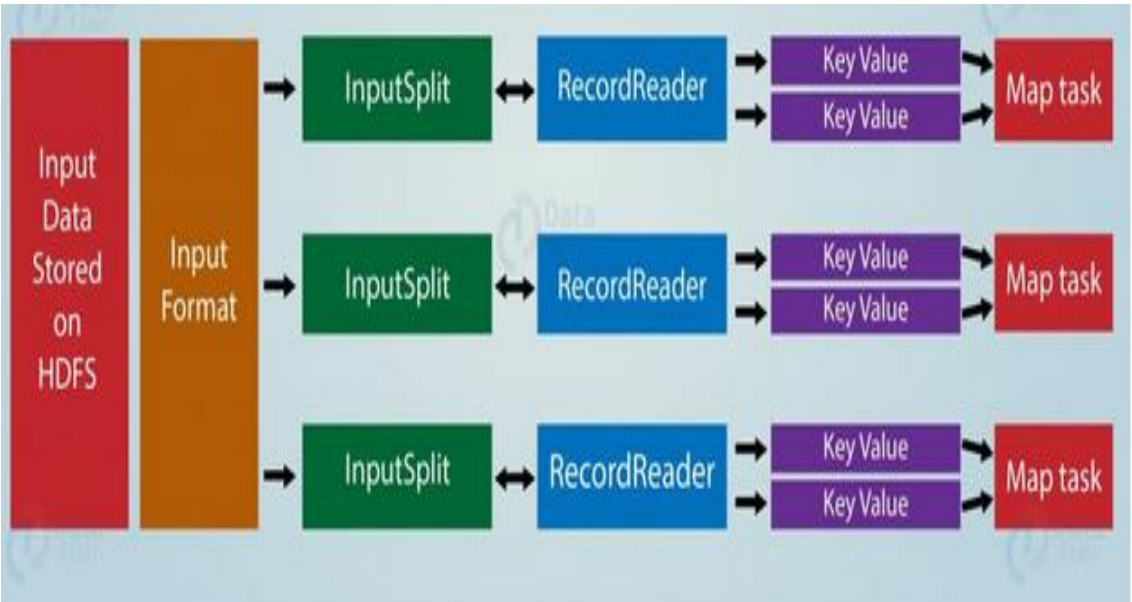

Figure 5. Key-value Pair Concepts in Hadoop MapReduce. (Learn the Concept of Key-Value Pair in Hadoop MapReduce by DataFlair Team, 2018).

In the MapReduce process , the data must first be converted to key-value pairs before having to pass the data to the mapper, because the mapper only understands key-value pairs of data. The value key pair is the object of record which the MapReduce job gets for execution. RecordReader uses TextInputFormat by default to translate information into a pair of key-values. InputSplit and RecordReader in Hadoop generate the key-value pairs. (Learn the Concept of Key-Value Pair in Hadoop MapReduce by DataFlair Team, 2018).

InputSplit: This is the logical description of the information. The data which the individual Mapper is to interpret shall be provided by the InputSplit.

RecordReader: This connects to the InputSplit and converts the Split into files in the form of key-value pairs suited to mapper learning. RecordReader uses TextInputFormat by default to translate information into a pair of key-values. RecordReader communicates with InputSplit until the processing of the file is over. (Learn the Concept of Key-Value Pair in Hadoop MapReduce by DataFlair Team, 2018).

2.5.2 MapReduce Execution Framework

MapReduce works by dividing the computation into two phases; Map Phase and Reduce Phase. For inputs and outputs each phase has key-value pairs. Besides that, two functions need to be specified, namely: a map function and a reduced function. (Hadoop Mapper–4 Steps Learning to MapReduce Mapper by DataFlair Team, 2018).

i. Map function: This is also known as the Mapper for Hadoop MapReduce. Mapper task occurs in the first processing stage, processing every RecordReader input record as well as generating an intermediate key-value pair. Hadoop Mapper saves the local disk for the intermediate data. Figure 6 illustrates the procedure of a typical Hadoop MapReduce Mappers. (Hadoop Mapper–4 Steps Learning to MapReduce Mapper by DataFlair Team, 2018).

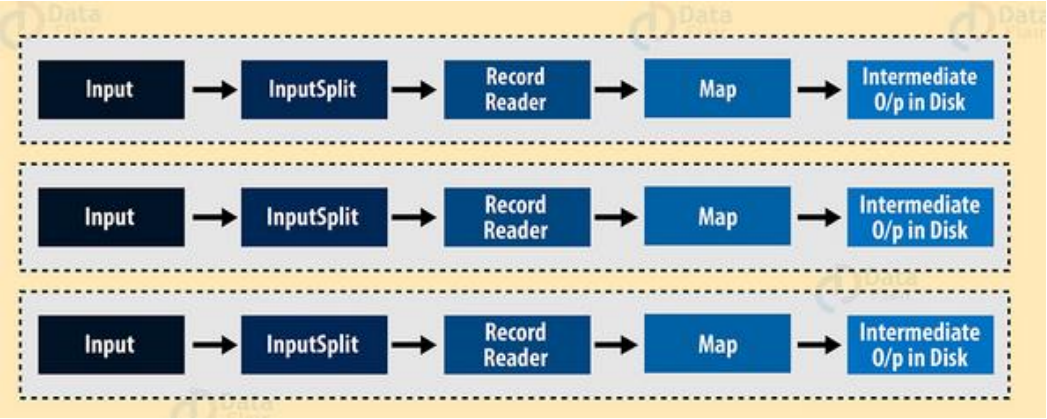

Figure 6. MapReduce Mapper (Hadoop Mapper–4 Steps Learning to MapReduce Mapper by DataFlair Team, 2018).

MapReduce Mapper processes each input record and generates new pairs (key, value). The pairs (key, value) could be completely different to the pair of inputs. The output is the complete set of all (key , value) pairs within the mapper function. Until the output for each mapper function is written, the output partitioning is done on the basis of the key, then sorting is done. This partitioning specifies grouping of all values for each key. For every InputSplit generated for the job by the InputFormat, MapReduce framework produces one map function. Mapper recognizes

only pairs of data (key, value) so data must be transformed into (key, value) first, before transferring data to the mapper. (Hadoop Mapper–4 Steps Learning to MapReduce Mapper by DataFlair Team, 2018).

ii. Reduced Function: The Reducer of Hadoop MapReduce, also known as Hadoop Reducer, takes the output of a Mapper process (intermediate key-value pair) to produce the output. The Reducer output is the final output which is stored in HDFS. The Reducer processes a mapper 's output, generates new output set, and stores the output data in HDFS. Figure 7 below shows the working principle of a Hadoop Reducer. (Hadoop Reducer–3 Steps learning for MapReduce Reducer by DataFlair Team, 2018).

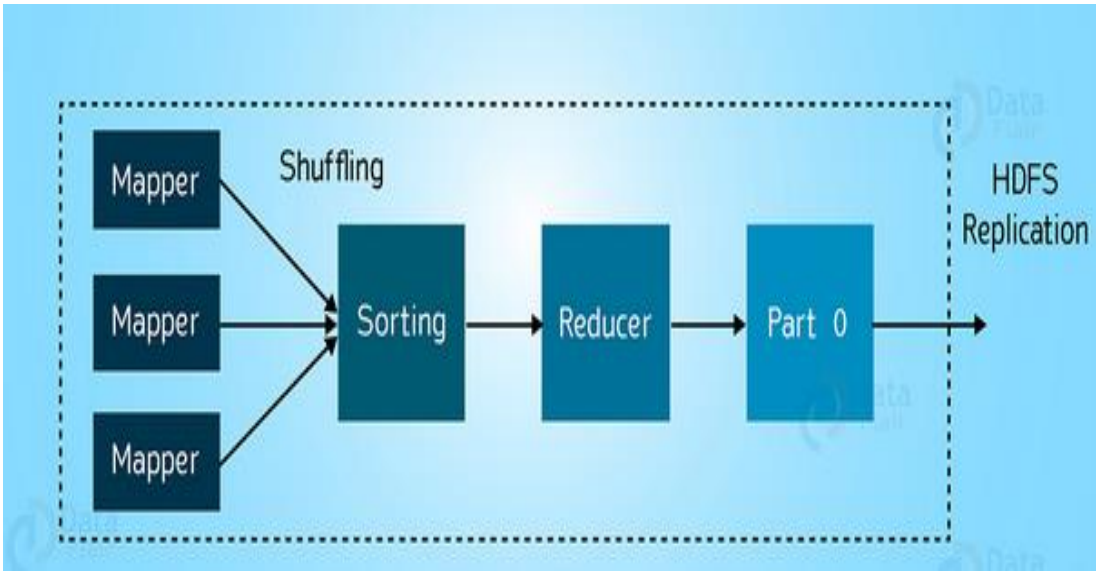

Figure 7. The working Principle of a Hadoop Reducer. (Hadoop Reducer–3 Steps learning for MapReduce Reducer by DataFlair Team, 2018).

It takes an input; a set of Mapper-generated intermediate key-value pairs, and runs the Reducer function on each of them. For a diverse array of processing operations, this data (key, value) can be aggregated, filtered and combined in several ways. The Reducer processes the intermediate

values generated by the Mapper function for the specific key and produces the output (zero or more key-value pairs). One-one mapping happens between the keys and the Reducers. The Reducer is running parallel, as they are independent from each other. The User can specify the number of reducers. (Hadoop Reducer–3 Steps learning for MapReduce Reducer by DataFlair Team, 2018).

2.6 Apache Spark

Apache Spark is a general distributed computing system built on Hadoop MapReduce algorithms. This incorporates the benefits of Hadoop MapReduce, however unlike MapReduce, the intermediate and output of Spark tasks are stored in memory called Memory Computing. The memory Computing increases information computing performance. Apache Spark is therefore suited for iterative applications such as data mining and machine learning. (J. Fu, et al., 2016).

Apache Spark is an open source computing engine that is extremely concerned with acceleration and reliability. It was built for fast computation. Spark is designed to operate in a Hadoop environment and to overcome the constraints of MapReduce. This system provides Application Programming Interface (APIs) in various programming languages such as Scala, Java and Python. Apache Spark is designed for real-time data processing and quick queries that end in a few seconds. (Aziz, et al., 2018)

2.6.1 Spark Architecture

Spark is centred around the RDD idea. RDD is a fault-tolerant array of components which can run in parallel and enables the user to store data directly on the disk and memory. It is a distributed computing model designed for large-scale, linearly scalable, and tolerant to fault. Carries out memory processing using the RDD data type.

During the storage process RDDs are a set of partitioned and permanent objects. Works are divided into a Master and several Slaves in Spark. The master assigns duties to the Slaves and gives the master the reports. The types of activities over the RDDs performed on Spark are classified into two categories: Transformations and Actions. The specification of these tasks is carried out in groups of transformations and executed as defined. This processing mode facilitates the absence of reloading data from the start of each transformation. This makes Spark very useful for iterative algorithms that involve multiple readings on a data set, as well as programs that require quick queries on large datasets. (Giraldo, et al., 2018).

I. Transformation: Transformations in Spark RDD are functions which accept the RDD as input and generate one or more RDDs as output. It does not change the input RDD because RDDs are immutable and therefore cannot be changed. However, produces one or more new RDDs by implementing the computations they represent. Transformations are considered lazy RDD operations in Apache Spark. It generates one or more additional RDDs that run when a process takes place. Consequently, Transformation creates a new dataset from the existing one. Other transformations can be carried out which is the optimization approach used by Apache Spark to improve the performance of the computation. (Introduction, Features & Operations of RDD, 2019).

II. Action: The Action operation in Spark returns the final result of the RDD computations. It activates execution using a lineage map to load the data into the initial RDD, execute all the intermediate transformations and send back the final results to the Driver program. The Lineage graph is the dependency graph of all RDD in parallel. The Actions operation are RDD operations that generate non-RDD values. The Action operation is one way to send the outcome from the executors to the driver. (The Hadoop Ecosystem: Welcome to the zoo!). Figure 8 shows a typical Spark Architecture

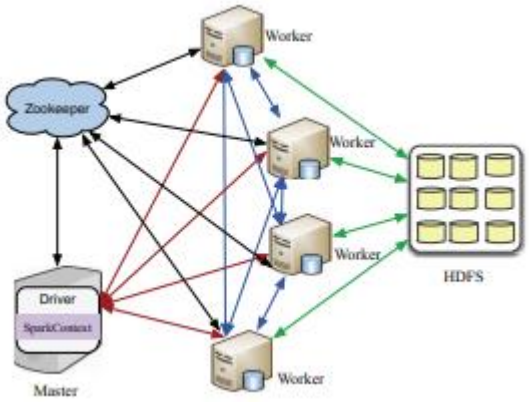

Figure 8. Spark Architecture. (X. Lu, et al., 2016).

Apache spark consists of the SparkContext, executors, cluster managers, and HDFS operating system. The primary software for spark is the driver program. Spark programs run as independent cluster system sets, controlled by the Driver program known as the SparkContext. Each application has its own processes and executed tasks in different threads, and the worker nodes need to be in the same network. Once paired, in the cluster that the worker processes, Spark ac-acquires node executors, then performs computation and stores data for your application. It then sends the application code, defined by JAR or Python files, passed to the executors through the SparkContext. Lastly, SparkContext sends tasks to run by the executors. (Verma, et al., 2016).

2.6.2 Solutions by Spark

The main reasons behind the RDD theory are iterative algorithms and interactive data mining tools. The Data from the distributed computing platform is stored in the intermediate secure data server, such as we have it in HDFS and Amazon S3. It makes computation slower, as it requires multiple Input/output operations and replications in the process. The following are the solutions provided by Apache Spark. (Introduction, Features & Operations of RDD by DataFlair Team, 2019).

I. In-Memory Computation: The Spark RDDs processes a statute for in-memory computing. Stores intermediate outcomes in distributed memory instead of stable storage.

II. Fault Tolerance: The Apache Spark RDDs are fault tolerant as they track data lineage information to automatically rebuild lost data in the case of a failure. It rebuilds lost data through failure using lineage. Each RDD knows how it was generated from other datasets.

III. Immutability: Data is safe to be distributed across systems. It can also be created or retrieved at any time that makes caching, sharing and replication simple. It is therefore a way to achieve reliability in computation.

IV. Partitioning: Partitioning is the conceptual model of parallelism in Apache Spark RDD. A partition is a single logical division of data that is mutable. A partition can be generated by some transformations on current partitions.

V. Persistence: The user can specify which RDDs to reuse and choose a storage strategy for it. (Introduction, Features & Operations of RDD by DataFlair Team, 2019).

3 THE CLOUD COMPUTING SERVICES

Cloud is held on Web servers rather than on the device. Moreover, when we use the cloud for computing, it implies that we take input for processing and send it to a client regarded as Cloud Computing. Cloud computing is an information technology system that works with the transfer of data, sharing of information, or services through a server across the Internet. It is the use of computing resources that are offered as a platform or an application across a network. It is supported by several of services, such as AWS and Google Cloud Platform. Cloud computing services can be divided into three categories namely: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Figure 9 shows a typical service model of a Cloud Computing. (Prajapati, et al., 2018).

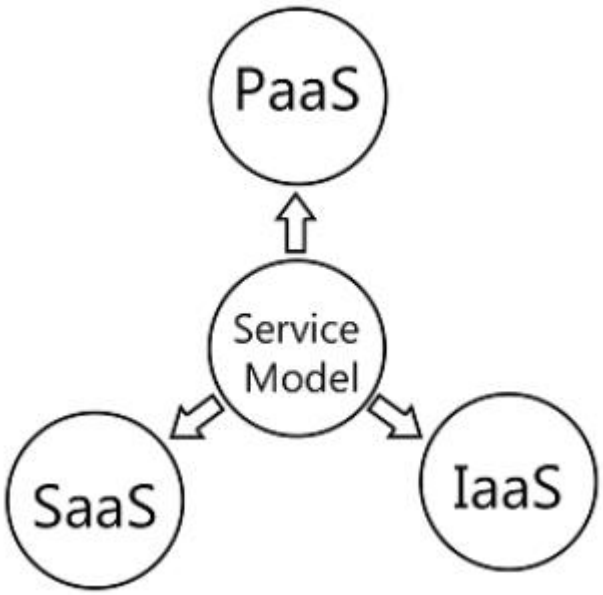

Figure 9. A Typical Cloud Computing Model. (Prajapati, et al., 2018).

I. IaaS: This provides virtualization, storage, and processing capabilities. We have no control over the cloud infrastructure in this model, but we can utilise and run software with this model. It is primarily used for data storage and product design e.g. AWS.

II. PaaS: It gives us a virtual development environment where users can build and install cloud applications We can not influence the cloud infrastructure, but we can control their implemented application e.g Microsoft Azure.

III. SaaS: This gives access to cloud-based application providers. The client does not have control over the cloud infrastructure, but has minimal control over the system settings. e.g. Drop Box. (Prajapati, et al., 2018).

3.1.1 Amazon Web Services

AWS provides far more capabilities and functionality within these platforms than any other cloud provider. This cloud platform provides over 165 fully featured applications, with no other large cloud provider providing more than 40 features. AWS provides services for a wide range of applications, including computing, storage, databases, networking, analytics, artificial intelligence and machine learning (AI), IoT, security and application development, deployment, and management. (Cloud computing with AWS).

AWS possesses the following for computation EC2, Elastic MapReduce, and Auto Scaling. It has Simple Storage Service (S3), Elastic Block Storage (EBS) and CloudFront for storage. Its network includes Virtual Private Cloud (VPC) and Elastic Load Balancing (ELB). (Zhou, et al., 2010).

Amazon EC2 provides more styles and sizes of computing instances, along with the most efficient GPU instances for machine learning. It also has more than twice as many database services, including both relational and non-relational database solutions. (Cloud computing with AWS).

Of the above services allows web-based computing by providing access to a well-established infrastructure that resides on thousands of computers. This gives costumers versatility to run their business on a web-based that is unrestrained by growth and de-

mand. Each of these services are capable of constructing an entire system of computational services. (Zhou, et al., 2010).

3.1.2 Amazon Elastic Compute Cloud

AWS provides far more capabilities and functionality within these platforms than any other cloud provider. This cloud platform provides over 165 fully featured applications, with over 40 features not provided by any other large cloud provider. AWS gives services for a wide range of applications including computing, storage, databases, networking, analytics, machine learning and Artificial Intelligence (AI), IoT, security, and application development, deployment and management. (Cloud computing with AWS).

AWS possesses the following for computation EC2, Elastic MapReduce and Auto Scaling. It has Simple Storage Service (S3), Elastic Block Storage (EBS) and CloudFront for storage. Its network includes Virtual Private Cloud (VPC) and Elastic Load Balancing (ELB). (Zhou, et al., 2010).

Amazon EC2 provides more styles and sizes of computing instances, along with the most efficient GPU instances for machine learning. It has more than twice as many database services, including both relational and non-relational database solutions. (Cloud computing with AWS).

Of the above services allows web-based computing by providing access to a well-established infrastructure that resides on thousands of computers. This gives costumers versatility to run their business on a web-based that is unrestrained by growth and demand. Each of these services are capable of constructing an entire system of computational services.

3.1.3 4The Linux Instances

The Linux instance is Amazon's root cause, with Elastic Block Store (EBS) volume. This means it is an Amazon backed by EBS. There is a choice to select the Availability Zone that an instance runs in or will automatically let an Amazon EC2 do. To ensure this, a key pair and a security group are specified when the instance is started. Upon connection of an Instance, the private key of the key pair must be specified during startup. Figure 10 illustrates the launching of an Instance. (Getting started with Amazon EC2 Linux instances).

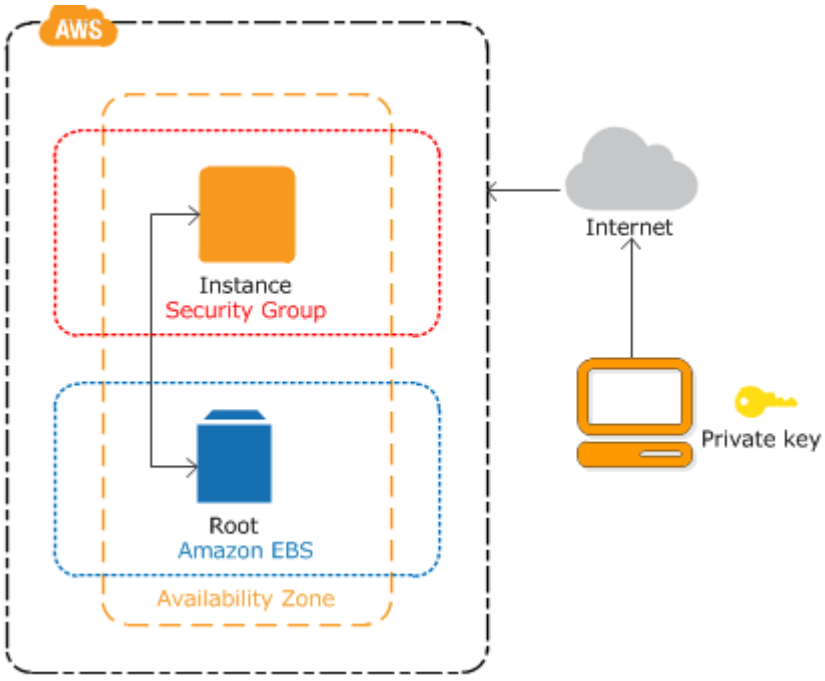

Figure 10. Lauching an Instance (Getting started with Amazon EC2 Linux instances).

3.2 Overview of Cloudera Manager

This is an end-to - end application for cluster management at CDH. It sets an enterprise deployment standard by providing concentrated visibility and control across all parts of the CDH cluster. It empowers operators to improve performance, the quality of service and minimize administrative costs. The Cloudera Manager makes the complete CDH stack and other managed services easy to deploy and manage centrally. CM automates the installation process, reducing downtime for deployment. It offers a cluster-wide, real time view of running hosts and services. It allows one single, central console for cluster-wide configuration changes. This incorporates a complete range of reporting and diagnostic tools that help in optimization of performance. Figure 11 illustrates the general concept of a CM. (Cloudera Manager Overview 6.3.x, Cloudera Enterprise, 2020).

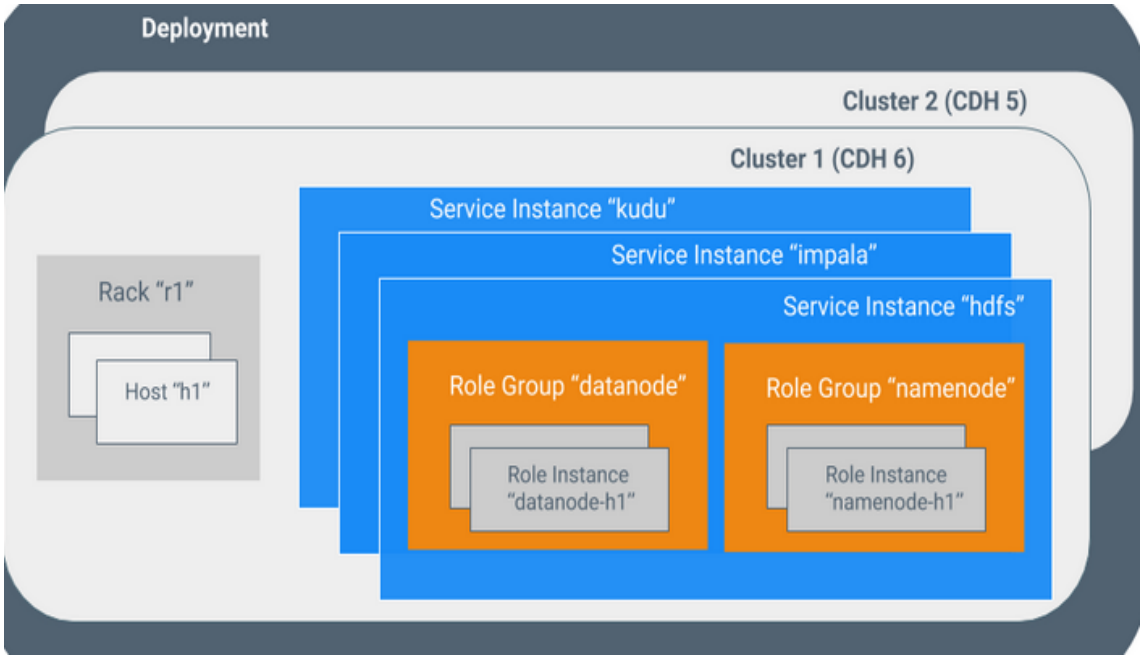

Figure 11. CM Overview (Cloudera Manager Overview 6.3.x, Cloudera Enterprise, 2020).

- Deployment: This is Cloudera administrator setup and all the clusters it handles.

- Rack: This is a physical object which consists of a set of physical hosts in CM operated by the same switch.

- Cluster: A collection of racks containing an HDFS filesystem running MapReduce and other tasks on the data.

- Service Instance: This is an instance of a service running on a cluster in CM. A service instance plays several roles of instances.

- Role Instance: A Role Instance is usually mapped to a Unix process in the CM. It is an instance of a role being played on a host.

- Role Group: This is a set the setup properties for a series of function instances.

3.3 The Cloudera Distribution Hadoop

This is an open source licensed under Apache. It is the only Hadoop implementation that provides integrated, interactive Structured Query Language (SQL) processing. This is the most extensive and popular Apache Hadoop distribution. It delivers Hadoop 's Key Components. It provides Web-based user interface for scalable storage and distributed computation. Figure 12 shows the key features of a CDH. (Cloudera Enterprise 5.14.x, CDH Overview, 2020).

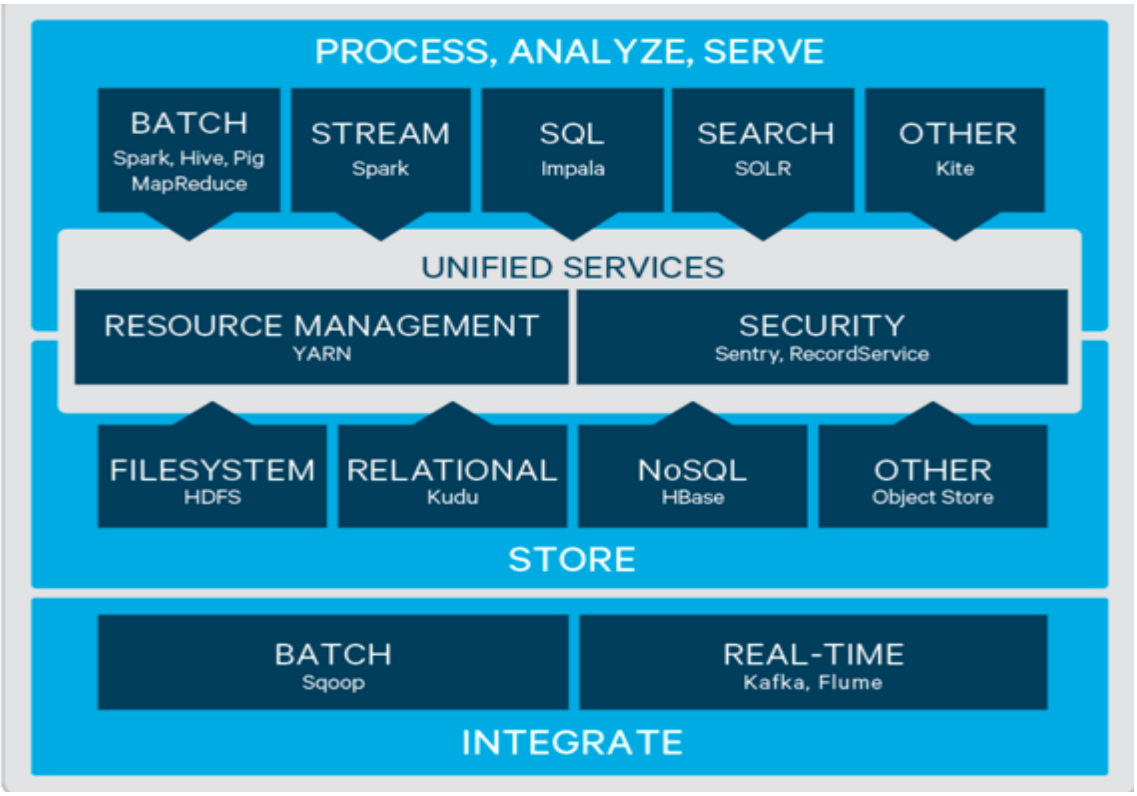

Figure 12. Features of a CDH. (Cloudera Enterprise 5.14.x, CDH Overview, 2020).

- Flexibility: It stores and manipulates any type of data with a variety of computing structures, interactive SQL and machine learning.

- Integration: It runs quickly on a full Hadoop platform that integrates with a wide range of hardware and software solutions.

- Scalability: It facilitates a wide range of applications and extend them to meet requirements.

- Compatibility: It utilizes the existing IT infrastructure.

3.4 Performance Metric Evaluation

The difference of a big-scale distributed computing environment is the elapsed time in a block processed of a gigabyte. When there are many machines working parallel to each other, the amount of time taken to process each block varies considerably. Sometimes blocks could have a lot of data to process, and this takes longer time than necessary. This variation in processing time is enormously important if it is correlated with the block value. The blocks with lots of data may, after all, have higher aggregate values and take longer to process. In the same way the nodes which process large blocks, these set of blocks that are more likely to take longer time. Tackling this in a distributed environment requires innovation in the design and statistical analysis of the systems. (NiketanPansare1, et al.).

3.4.1 The Cluster Throughput

This is the amount of job that can be done in each timeframe. It is measured in seconds / bytes.

3.4.2 The Elapsed Time

This is the difference between start time and end time for a job completion. It is the time it takes to perform an event. The performance can be differentiated as less time elapsed indicates an effective performance. When the average timing of job completion is reduced, there is an increase of the overall effective throughput. Its measurement is in seconds.

4 IMPLEMENTATION AND BENCHMARKING

The aim of this implementation is to design a four-node cluster where one node acts master and the rest as slaves. This design is considered because we want the master to coordinate the scheduling and management of all the slaves with high performance. This also ensures fault tolerance and data recovery in case one of the slaves malfunctioned for reliable benchmarking. For a two-node cluster, the data and associated workloads can only be shifted from one pair to the other. However, with a four-node cluster, the workload can be migrated within any of the three nodes. The application of this is that performance and data storage capacity is distributed across the cluster at a given composition with more flexibility.

This implementation was carried out using AWS. An instance represents a node; I set up four Linux instances and connected them using SSH. The four Linux instances were assigned a static Internet Protocol (IP). The virtual IP was configured using Amazon Virtual Private Cloud (VPC). The VPC enables the lunching of Linux instances into a defined virtual network. These four Linux instances from the four cluster nodes. The instances details were configured using the hostname. The CDH was now deployed on the clusters.

4.1 Configuration of Virtual Private Cloud

VPC allows the launch of AWS resources into a pre-defined virtual network. VPC scalable infrastructure is the networking layer for Amazon EC2. The created single subnet VPC, as shown in Figure 13.

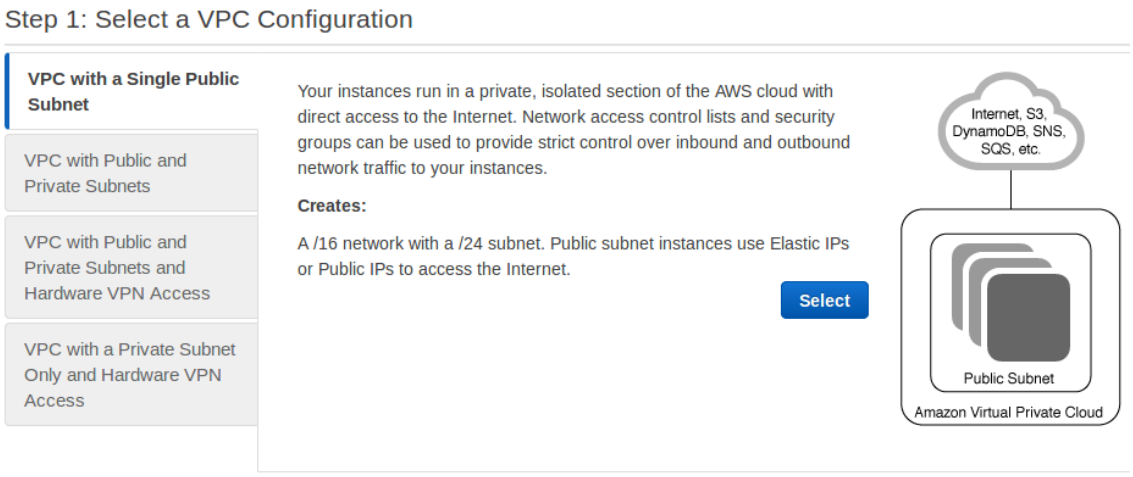

Figure 13. Starting VPC Configuration

From the Amazon VPC dashboard, the VPC with a Single Public Subnet has been launched as shown in Figure 14.

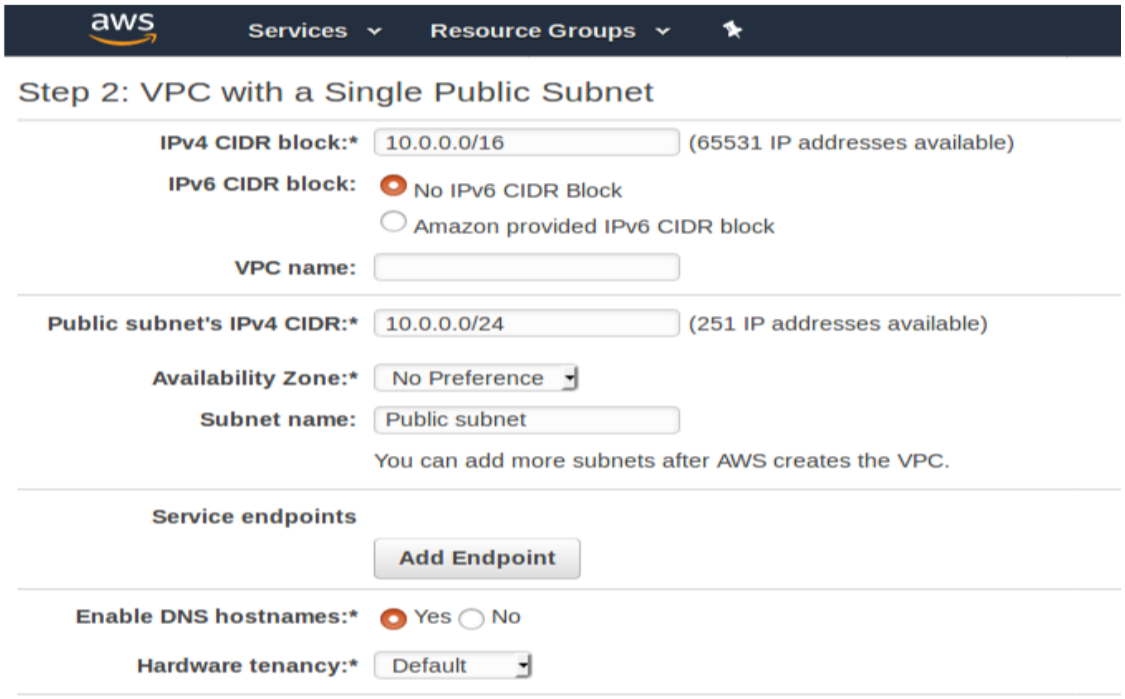

Figure 14. Single Subnet VPC Configuration

The IPv4 CIDR block shows the IPv4 address span and that of IPv4 subnet too. No preference Availability Zone has been selected because we want it to be selected by AWS. We enabled DNS hostnames so that launched Linux Instances with our VPC network get a DNS hostname.

4.2 Setting Up Security Group

A security group with its associated instances acts as a virtual firewall for controlling the traffic. An inbound rule for incoming traffic, and the outbound rules for the out-going traffic is specified. The Security Group Configuration is shown in Figure 15.

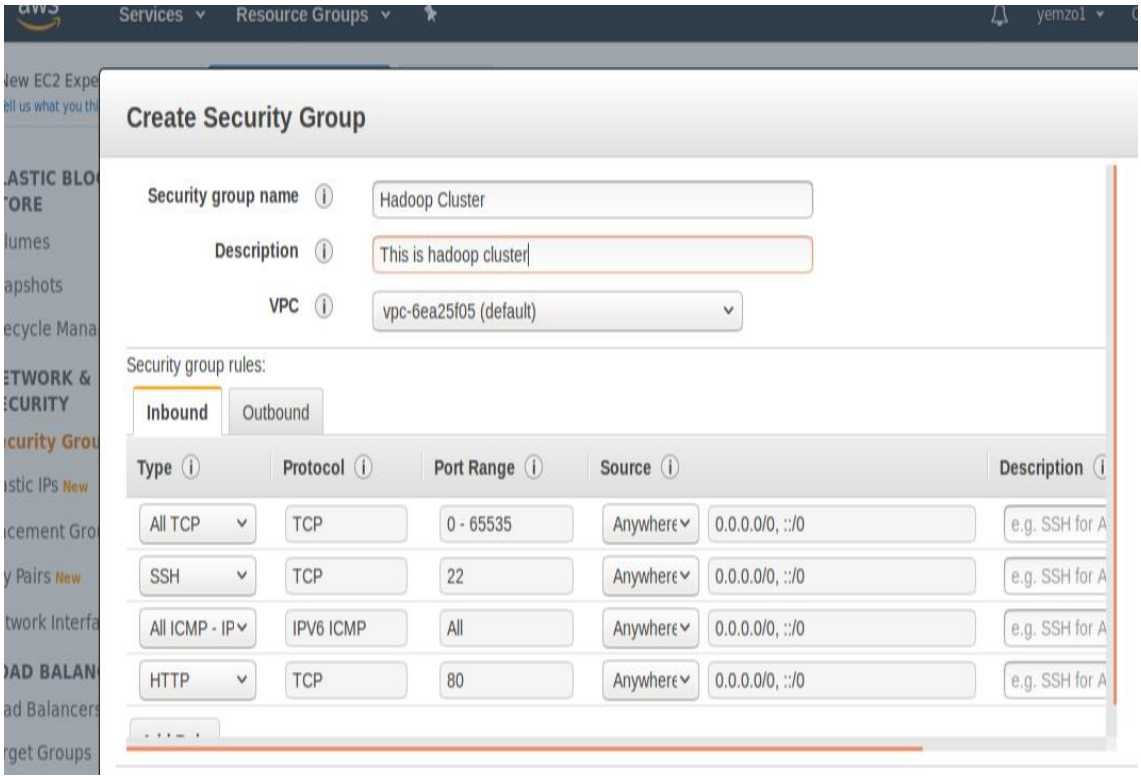

Figure 15. Security Group Configuration

Security group was selected from the Amazon VPC console. The group and description name was given as shown above in Figure 15. The created VPC was selected from the

VPC menu. From the inbound rules tab, we added rules for inbound traffic. The SSH was selected because the rules were for the lauching of Linux Instance. I configured the source field for HTTP and HTTPS to be 0.0.0.0/0 because we want to enable all the IP addresses via SSH.

4.3 Creating Key Pairs

The Amazon EC2 utilizes 2048-bit SSH-2 RSA keys. It stores the public key while the user is saving the private key. It uses public key cryptography to encrypt and de-crypt information about the logins. The key pair are known as both private and public keys. Public key cryptography allows the secure use of a private key, rather than a password, to access instances. Therefore, the private keys should be in a safe location.

The name of a key pair is requested before launching an instance. Using SSH for instance connection requires a key pair. At boot time, on your Linux instance, the public key content is placed in an entry within ~/.ssh/permitted keys. Since we used the SSH to connect to the Linux instances, it is necessary to specify the private key corresponding to the public key content in others to log in. Key Pairs is selected from the Amazon EC2 navigation panel as shown in Figure 16. We saved the private key name as **pem file** format because it will be used with **OpenSSH.**

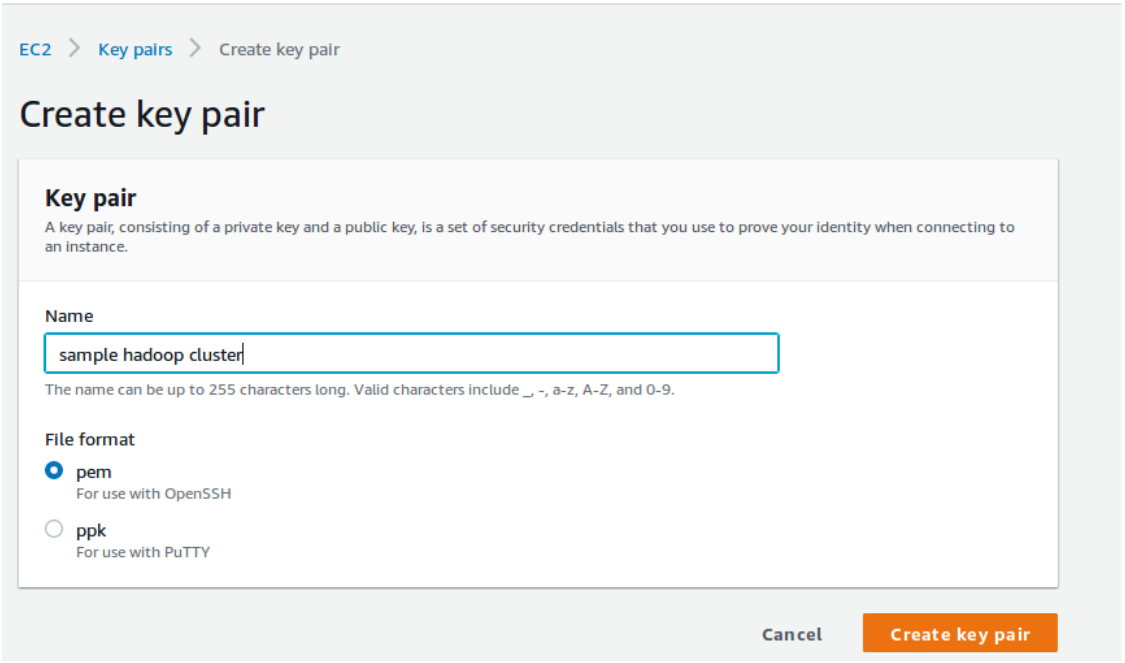

Figure 16. The Creation of Key Pair.

4.4 Configuring Linux Instances

We launched the Linux Instances into the VPC created. From the Amazon EC2 console dashboard, a free tier eligible AMI was selected. A t2.micro type instance was chosen because it is eligible for free tier as shown in Figure 17.

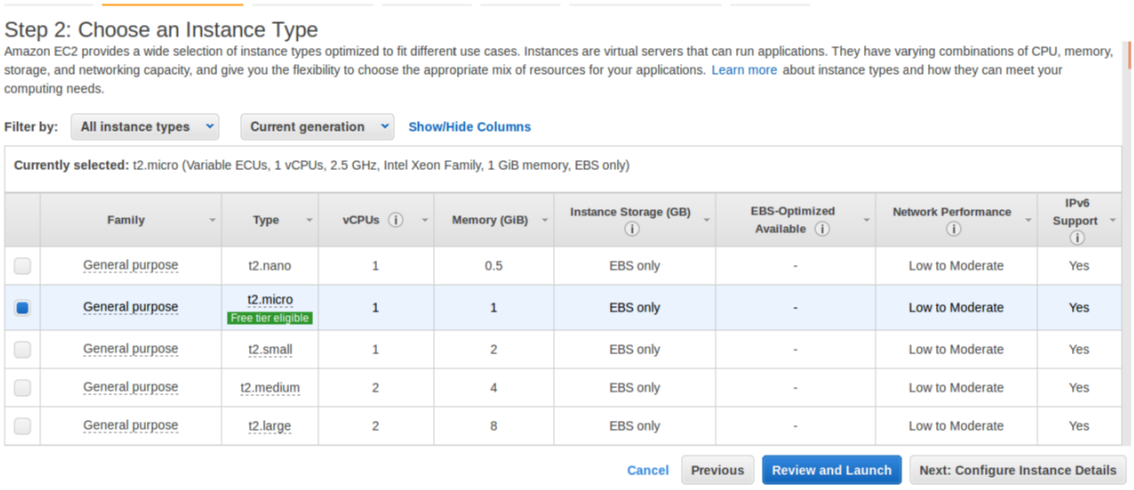

Figure 17. Lunching an Instance Type.

The Linux Instances details were configured as shown in Figure 18. The number of Linux Instances were selected to be four. The VPC created was selected as the network and Public Subnet chosen. Auto-assign public IP was enabled and reservation capacity left opened.

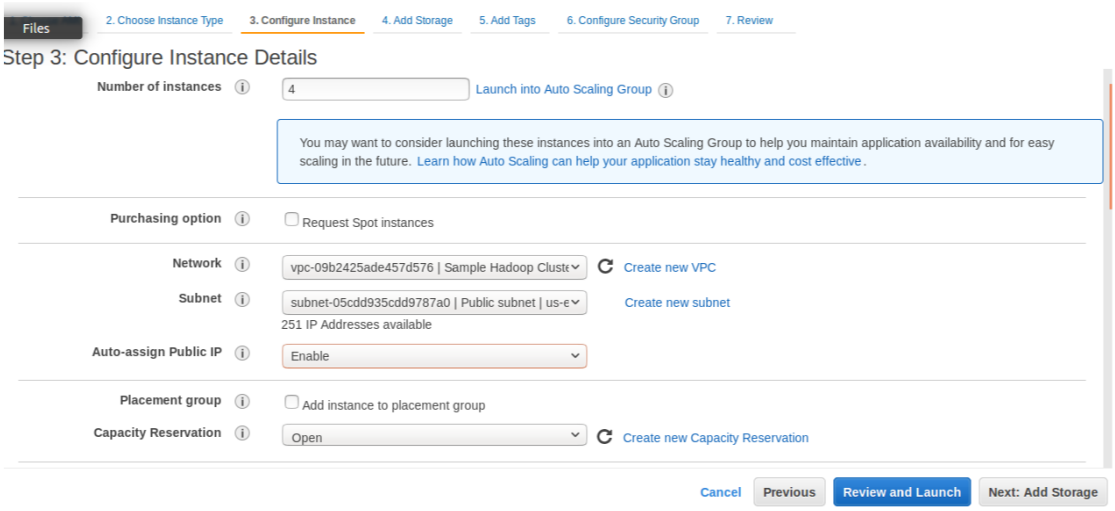

Figure 18. Configuration of Linux Instance Details.

A name tag was added to each of the Instances, so that each of them can be easily identified in the Amazon EC2 console after lauching. The configured security group is

then chosen from **Select an existing security group** option. The created security group was then selected as shown in Figure 19.

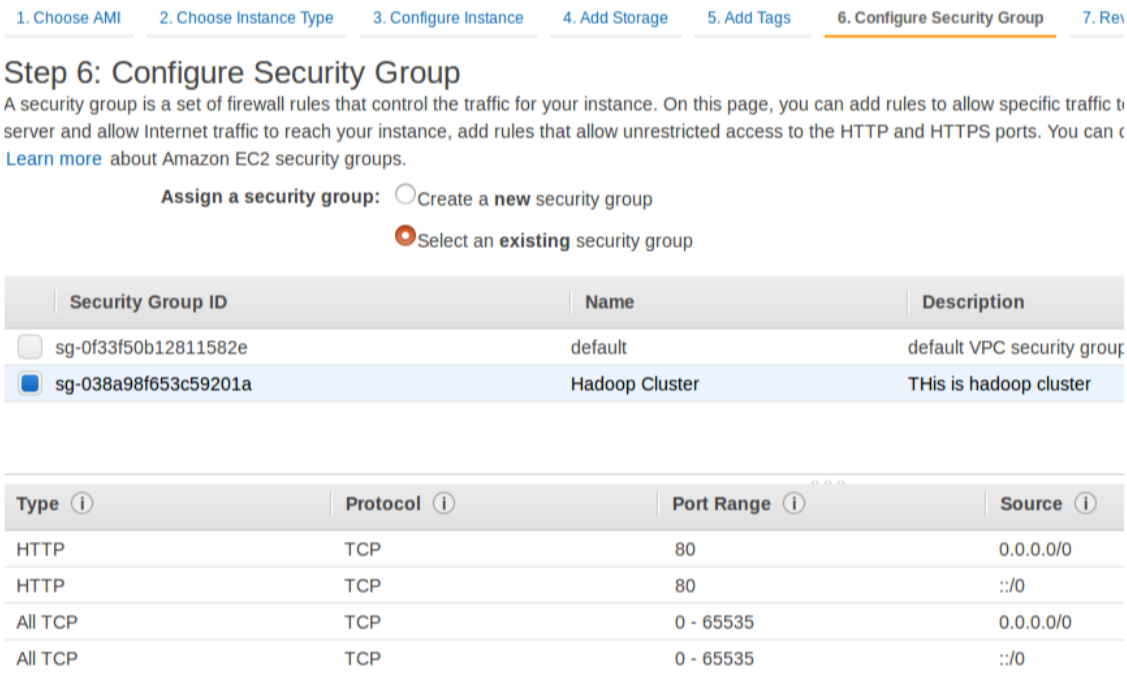

Figure 19. Configuration of Security Group

The created **key pair** was then selected, and the Linux Instances launched. From the confirmation page, the instances view displays as shown in Figure 20.

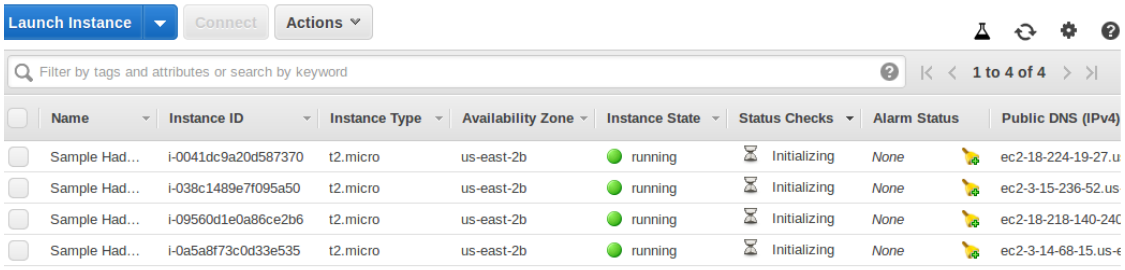

Figure 20. View of Linux Instances after Launching

4.5 Assigning Elastic IP Addresses

Previously launched Linux Instances were configured to form a public subnet. A subnet that has a gateway to its route over Internet. Public IPv4 address is also needed for the Linux Instances to be able to communicate with the Internet. By default, a pub-lic IPv4 address is not assigned to a Linux Instance VPC. Our account has been assigned an Elastic IP address and is then associated with the Linux instances configured as shown in Figure 21.

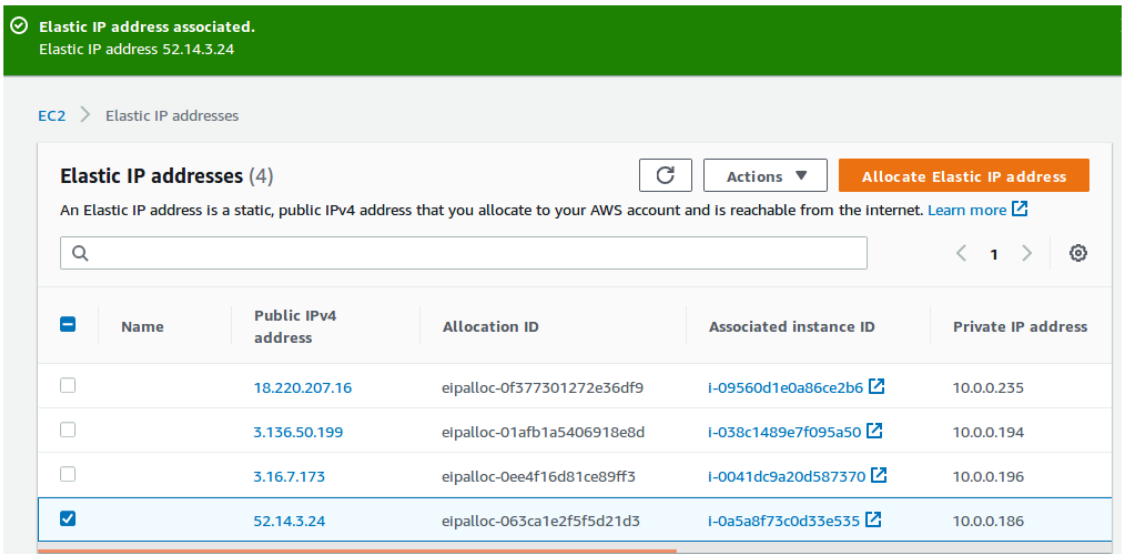

Figure 21. Associating Elastic IP addresses to Linux Instances

4.6 Connecting Linux Instances

The connection to the Linux Instances were established using SSH clients. The SSH command was used from the terminal with specified path and the private key file name, AMI username, and the Linux IP address or DNS name. The following commands were used connecting the Linux Instances.

```
yemzo@yemzo-Lenovo-E31-80:~$ cd aws_adeyemi/

yemzo@yemzo-Lenovo-E31-80:~/aws_adeyemi$ ssh -i samplehadoopclus-
ter.pem ubuntu@ec2-3-16-7-173.us-east-2.compute.amazonaws.com
```

There comes up an RSA authentication that needed to be accepted as follows.

```
RSA key fingerprint is
1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f.Are you
sure you want to continue connecting (yes/no)
```

Once “yes” is entered comes to the information below to show a successful connection is established to the master.

```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1051-aws x86_64)

* Documentation:  https://help.ubuntu.com

 * Management:     https://landscape.canonical.com

 * Support:        https://ubuntu.com/advantage

System information as of Wed Dec 25 15:45:35 UTC 2019

 System load:  0.0               Processes:           99

  Usage of /:   38.5% of 7.69GB   Users logged in:     1

  Memory usage: 20%               IP address for eth0: 10.0.0.196
```

```
  Swap usage:   0%

* Overheard at KubeCon: "microk8s.status just blew my mind".

https://microk8s.io/docs/commands#microk8s.status

* Canonical Livepatch is available for installation.

   - Reduce system reboots and improve kernel security. Activate at:

     https://ubuntu.com/livepatch

37 packages can be updated.

0 updates are security updates.
```

The Private IP addresses were resolved to Master, Slave01, Slave02 and Slave03 as shown below.

```
10.0.0.196 ec2-3-16-7-173.us-east-2.compute.amazonaws.com      master

10.0.0.194 ec2-3-136-50-199.us-east-2.compute.amazonaws.com    slave01

10.0.0.235 ec2-18-220-207-16.us-east-2.compute.amazonaws.com   slave02

10.0.0.186 ec2-52-14-3-24.us-east-2.compute.amazonaws.com      slave03
```

The allocated Elastic Public IPv4 addresses to the Master and Slaves are as shown below:

```
3.16.7.173      master

3.136.50.199    slave01
```

```
18.220.207.16   slave02

52.14.3.24      slave03
```

The essence of the Elastic IP addresses is to prevent automatic assigned of IP addresses each time the Linux Instances are launched. Since the Master must have undeniable access to all the Slaves for the formation of four clusters. The Slaves fingerprint are copied into the Master as described below for Slave03.

```
ubuntu@ip-10-0-0-196:~$ nano /home/.ssh/id_rsa.pub

ubuntu@ip-10-0-0-196:~$ cat /home/ubuntu/.ssh/id_rsa.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDGl0jmh/Tf7A6ijPygnXSfZmH0Aw5SSW/6VO8Lt1
sWIGbWHtR4qS2icrFxcrV3XZfIkmq7QRl+Tek7dGjsH+UolWJYdVNU8yAnnyIkLYbleLgm
LABKM+CGBR4uWLEt9i8ooh3lAyeRhkz/JMKKbuT3kzJNbAYQC81mGlT1m4KiOSpvIo4iTB
Zshi7P1YTt3U7qEgjbSC3Vp7HubMiac/EUd6q+5N7f4DX0yFdP96eh471BsibCcdRWA2VN
b+D1NxEq71+r8cbIG8Q7R9r9CeQrZNA/FwqtsMGkrwHCWWyDCKKOtFaiypt1KQOfYtn8Ko
U5chrdQN8QU3tB/bqPCymD ubuntu@ip-10-0-0-196

ubuntu@ip-10-0-0-196:~$ ssh slave03

The authenticity of host 'slave03 (52.14.3.24)' can't be established.

ECDSA                key                fingerprint                is
SHA256:uTo42XMlTrLkWgrHOVjjlthSRrd5GHbDaBpTsL89fhU.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'slave03,52.14.3.24' (ECDSA) to the list of
known hosts.

Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1051-aws x86_64)
```

```
* Documentation:  https://help.ubuntu.com

 * Management:     https://landscape.canonical.com

 * Support:        https://ubuntu.com/advantage

   System information as of Wed Dec 25 15:53:51 UTC 2019

   System load:  0.0               Processes:           91

  Usage of /:   20.0% of 7.69GB   Users logged in:     1

  Memory usage: 19%               IP address for eth0: 10.0.0.186

  Swap usage:   0%

* Overheard at KubeCon: "microk8s.status just blew my mind".

     https://microk8s.io/docs/commands#microk8s.status

* Canonical Livepatch is available for installation.

   - Reduce system reboots and improve kernel security. Activate at:

     https://ubuntu.com/livepatch

39 packages can be updated.

0 updates are security updates.
```

4.7 Cloudera Manager and Cluster Installation

The cloudera manager is installed with the aid of package tools wget. The installation is described below.

```
ubuntu@ip-10-0-0-196:~$wget
https://archive.cloudera.com/cm6/6.3.1/cloudera-manager-installer.bin

--2020-01-03                                                  13:43:57--
https://archive.cloudera.com/cm6/6.3.1/cloudera-manager-installer.bin

Resolving        archive.cloudera.com        (archive.cloudera.com)...
151.101.248.167

Connecting           to           archive.cloudera.com           (ar-
chive.cloudera.com)|151.101.248.167|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 836826 (817K) [application/octet-stream]

Saving to: 'cloudera-manager-installer.bin'

cloudera-manager-in 100%[==================>] 817.21K  2.62MB/s    in
0.3s

2020-01-03  13:43:58  (2.62  MB/s)  -  'cloudera-manager-installer.bin'
saved [836826/836826]
```

After the installation, the cloudera manager is opened into the web using the master's IP address with port 7180. The default login details are admin for username and password. All the hosts names are specified, added and the cluster installation begins. The cloudera manager and JDK packages were updated on all the nodes. The CDH is now downloaded, updated, distributed, and activated all the packages. The services such as

HDFS, HIVE, YARN, ZooKeeper, and Hue was all selected and added to the cluster as shown in Figure 22a.

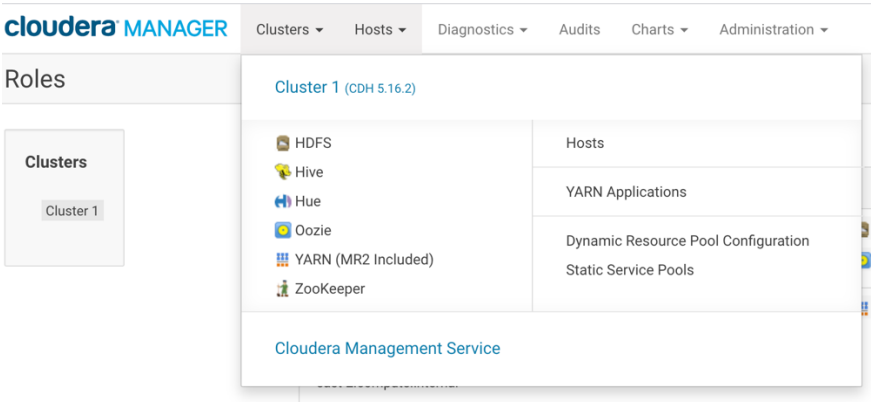

Figure 22a. The Cluster with Services

Service roles were also added to the host's i.e, the master and slaves. Figure 22b shows the cluster setup after installation with added services.

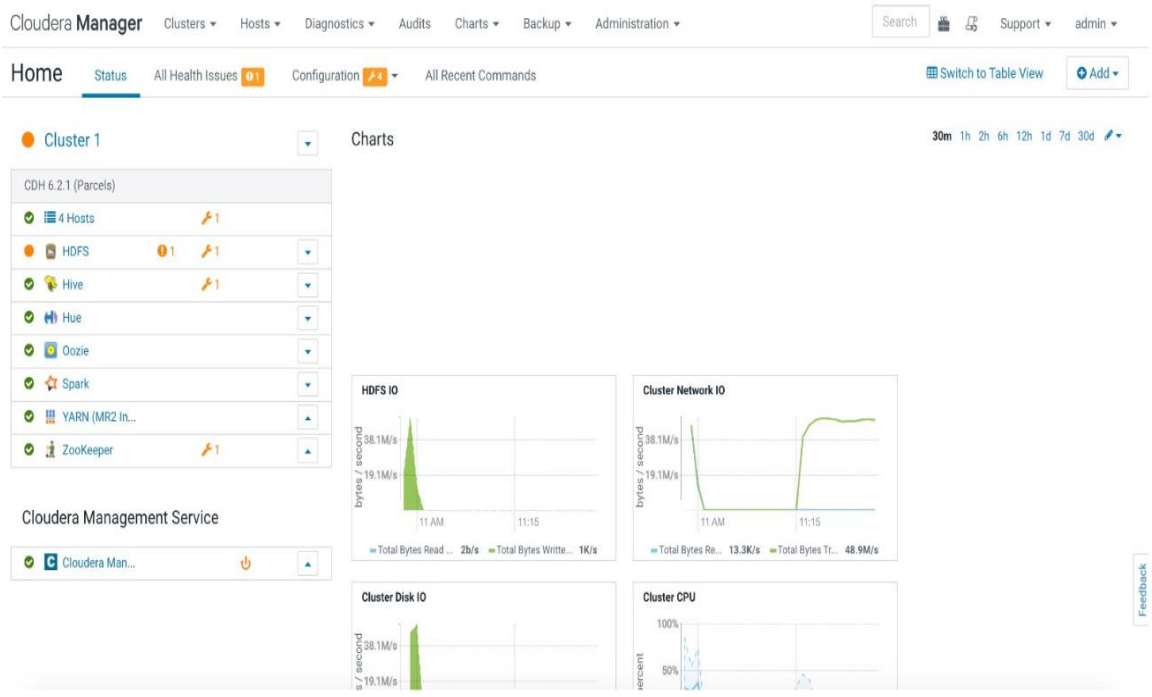

Figure 22b. The cloudera Web Page After Cluster Installation

4.8 Benchmarking

The purpose of this benchmarking is to subject both MapReduce and Apache Spark to the same relative amount of task and observe their performances. The tasks have been categorized into queries from the simplest to the most difficult. There are four different types of queries to run the benchmark, namely:

i. Scan Query: The MapReduce and Spark jobs will fully scan the table to retrieve the contents or data as stated in the Scan jobs. It involves the data processing engines, Hadoop MapReduce and Spark, to scan data in order to find a particular record satisfying a given criteria.

ii. Aggregate Query: The MapReduce and Spark jobs will retrieve data by analysing set of data entries. The aggregate query groups record together using a specified attribute, it then provides a summary of this group.

iii. Two-Way Join Query: The MapReduce and Spark jobs will retrieve data from two tables and join them as a single set of data. The two-way join query involves combining attributes from two relations and then performing aggregation on the combined data according to a specified grouping.

iv. Three-Way Join Query: The MapReduce and Spark jobs will retrieve data from three tables and join them as a single set of data. Our three-way join query combines attributes from three relations (books, revenue, transaction), and then performs an aggregation on the combined data according to a specified grouping.

The Big Datasets was generated using a data generator, Amazon Elastic MapReduce data generation tool that generates data files for three entities books, customers, and transactions. Desired data sizes of entities to be generated can be specified in the com-

mand line. Big data was generated, and directory created. The generated data was saved into the created directory. The data is then moved into HDFS as described below. (Amazon Elastic MapReduce: Developer Guide)

```
# we will generate 5gb data for following entities books, customers
#    and transactions, and save into the directory benchmark/data

# create the directories to save data into
mkdir -p benchmark/data

# generate data and specify directory created above
java -cp dbgen-1.0-jar-with-dependencies.jar DBGen -p benchmark/data -b 5 -c 5 -t 5

# we create a directory in HDFS and copy the generated data into this #    directory using the
mkdir and copyFromLocal commands

hadoop fs -mkdir /user/yemi/benchmark/data/
hadoop fs - copyFromLocal benchmark/* /user/yemi/benchmark
```

The Hive database and Tables were created over HDFS data as described and shown in Figure 23.

 # Create Hive database and Tables over HDFS data

```
CREATE SCHEMA benchmark_db;
USE benchmark_db;
CREATE EXTERNAL TABLE books(
    bk_id BIGINT,
    bk_isbn STRING,
    bk_category STRING,
    bk_publish_dt TIMESTAMP,
    bk_publisher STRING,
    bk_price FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LOCATION '/user/yemi/benchmark/data/books/';


create external table customers(
    cus_id BIGINT,
    cus_name STRING,
    cus_dob TIMESTAMP,
    cus_gender STRING,
    cus_state STRING,
    cus_email STRING,
    cus_phone STRING )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LOCATION '/user/yemi/benchmark/data/customers/';


create external table transactions(
    trans_id BIGINT,
    trans_customer_id BIGINT,
    trans_book_id BIGINT,
    trans_quantity INT,
    trans_transaction_dt TIMESTAMP )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LOCATION '/user/yemi/benchmark/data/transactions/';
```

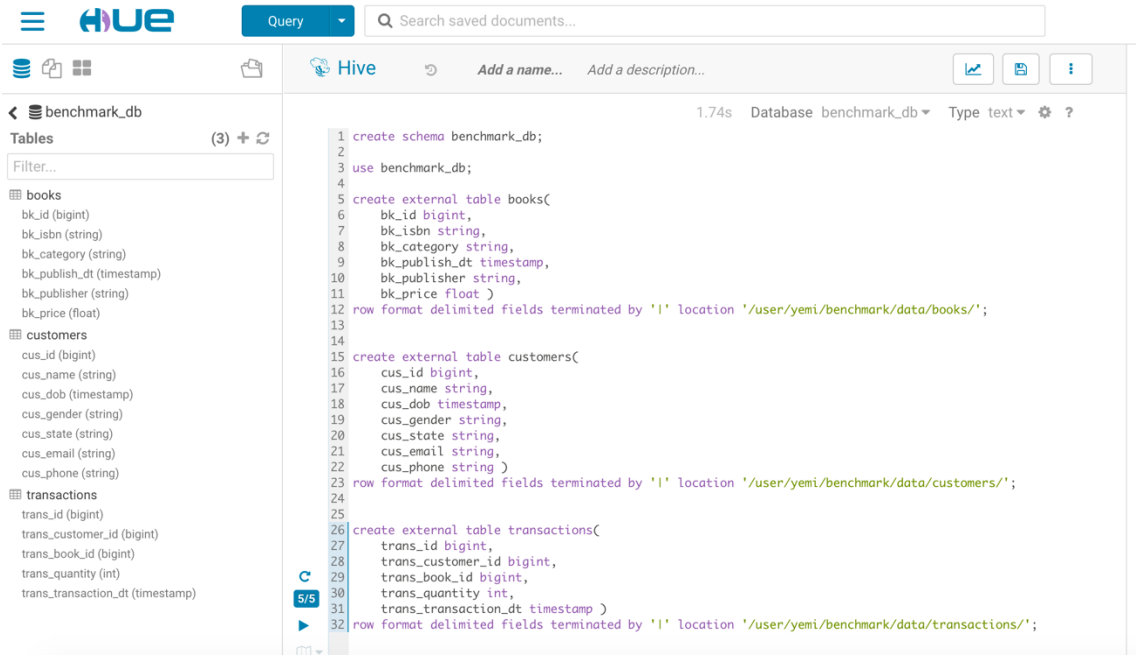

Figure 23. Hive Table Creation.

The MapReduce Job: It runs Hive queries over the table and measures the time the MapReduce jobs take.

The Spark Job: The spark job reads the same data from HDFS and runs the query using Spark APIs, as Spark dataset. It runs each of the classes by submitting them as spark jobs to Yarn, which is a resource manager and the times taken measured.

4.8.1 Scenario 1- Scan Query

HIVE query for MapReduce jobs is run over the table for total number of books in category “TECHNOLOGY ENGINEERING” and the execution time recorded i.e. Find books belonging to category Technology-Engineering.

 # MapReduce Job: Scan query

```
SELECT Count(*)
FROM books
WHERE bk_category IN ('TECHNOLOGY-ENGINEERING');
```

The Spark Scan job reads the same data from the HDFS and run the query with Spark APIs for total number of books in category “Technology Engineering”, as Spark dataset. The execution time is recorded.

 # Spark Job(Equivalent query in Spark API)

```
//case classes for books
case class Book(bk_id:String, bk_isbn:String, bk_category:String, bk_publish_dt:String,
bk_publisher:String, bk_price: String)

//get column
val columnNames = classOf[Book].getDeclaredFields.map(x => x.getName)

//read data as Spark Dataframe
val data = spark.read.option("header", false).option("delimiter","|").csv(hdfsFilePath +
"/books/books")).toDF(columnNames:_*).as[Book]

//filter dataframe
val techBooks = data.filter(_.bk_category == "TECHNOLOGY-ENGINEERING")
println(s"count: ${techBooks.count()}")
```

4.8.2 Scenario 2- Aggregation Query

The scenario is to get the total number of books in an orderly manner in ascending orderand with a limit of 10 per publisher. i.e. Find the first 10 Publishers with the largest number of books, display them from highest to lowest.

 # MapReduce Job

```
SELECT bk_publisher, Count(*)cnt
FROM books
GROUP BY bk_publisher
ORDER BY bk_publisher DESC LIMIT 10;
```

This scenario is to get the total number of books in an orderly manner by publisher using Spark API in ascending order and with a limit of 10 per category i.e. Find the first 10 Publishers with the largest number of books, display them from highest to lowest.

 # Spark Job (Equivalent query in Spark API)

```
//case classes for books
case class Book(bk_id:String, bk_isbn:String, bk_category:String, bk_publish_dt:String,
bk_publisher:String, bk_price: String)

//get column names
val columnNames = classOf[Book].getDeclaredFields.map(x => x.getName)

//read data as Spark dataframe
val data = spark.read.option("header", false).option("delimiter","|")
.csv(hdfsFilePath + "/books/books").toDF(columnNames:_*).as[Book]

//Aggregate data according to condition
val aggregatedData = data.groupBy("bk_publisher")
  .agg(count("*").alias("cnt"))
  .orderBy(desc("cnt"))
  .limit(10)
aggregatedData.show()
```

4.8.3 Scenario 3- Two Way Join Query

This scenario is to get the total number of books by revenue generated between 2008 and 2010 from the books and revenue columns in a descending order by category and 10 per each category i.e. For books bought between 2008 and 2010, find the first 10 Publishers with the highest sales. Display them from highest to lowest.

 # MapReduce Job

```
WITH a AS
(
         SELECT   b.bk_publisher                       AS publisher,
                  Sum(b.bk_price * t. trans_quantity) AS sum_sales
         FROM     transactions t
         JOIN     books b
         ON       t.trans_book_id = b.bk_id
         AND      Year(t.trans_transaction_dt) BETWEEN 2008 AND 2010
         GROUP BY b.bk_publisher )
SELECT   a.publisher,
         Round(a.sum_sales, 2) AS total_sales
FROM     a
ORDER BY total_sales DESC limit 10;
```

This scenario is to get the total number of books by revenue generated between 2008 and 2010 from the books and revenue columns in a descending order by category and 10 per each category using Spark and API query.

 # Spark Job (Equivalent query in Spark API)

```
//case classes for the two entities
case class Book(bk_id:String, bk_isbn:String, bk_category:String, bk_publish_dt:String, bk_publisher:String,
bk_price: String)
case class Transaction(trans_id:String , trans_customer_id :String, trans_book_id :String, trans_quantity :String,
trans_transaction_dt :String)

//get column names
val columnNamesBook = classOf[Book].getDeclaredFields.map(x => x.getName)
val columnNamesTransaction = clas-sOf[Transaction].getDeclaredFields.map(x => x.getName)

//Read data as Spark Dataframes
val book = spark.read.option("header", false).option("delimiter","|")
  .csv(hdfsFilePath  + "/books/books").toDF(columnNamesBook:_*).as[Book]
val transaction = spark.read.option("header", false).option("delimiter","|")
  .csv(hdfsFilePath  + "/transactions/transactions").toDF(columnNamesTransaction:_*).as[Transaction]

//Join book and transactions dataframes together and filter based on conditions
val data = book
  .withColumnRenamed("bk_publisher","book_publisher")
  .withColumnRenamed("bk_price","book_price")
  .join(transaction
    .withColumnRenamed("trans_id","transaction_id")
  )
  .where($"trans_id" === $"book_id")
  .filter(year($"trans_transaction_dt".cast(TimestampType)).between("2008", "2010"))
  .select($"book_publisher", $"book_price".cast(DoubleType), $"trans_quantity".cast(IntegerType))
  .groupBy("book_publisher")
  .agg(round(sum($"book_price" * $"trans_quantity"),2).alias("total_sales"))
  .limit(10)
data.show()
```

4.8.4 Scenario 4-Three Way Join Query

This is to get to get the list of books by price, revenue, and quantity with a limit of 10 per each category. Customer's state was also included a newly created column for the output i.e. For books bought by Female customers in NY, CA, LA and WA, find the first 10 Publishers with the highest sales. Display them from highest to lowest

 # MapReduce Job

```
WITH a
     AS (SELECT b.bk_publisher                      AS publisher,
                Sum(b.bk_price * t. trans_quantity) AS sum_sales
         FROM   transactions t
                JOIN books b
                  ON t.trans_book_id = b.bk_id
                JOIN customers c
                 ON t.trans_customer_id = c.cus_id
                     AND c.cus_gender IN ('F')
                     AND c.cus_state IN ('NY', 'CA', 'LA', 'WA')
         GROUP BY b.bk_publisher)
SELECT a.publisher,
       Round(a.sum_sales, 2) AS total_sales
FROM   a
ORDER BY sum_sales
```

This is to get to get the list of books bought by Female customers in NY, CA, LA and WA, find the first 10 Publishers with the highest sales. Display them from highest to lowest. Customer's state was also included a newly created column for the output using Spark API.

 # Spark Job (Equivalent query in Spark API)

```
//case classes for the two entities
case class Book(bk_id:String, bk_isbn:String, bk_category:String, bk_publish_dt:String,
bk_publisher:String, bk_price: String)
case class Transaction(trans_id:String , trans_customer_id :String, trans_book_id :String, trans_quantity
:String, trans_transaction_dt :String)

//get column names
val columnNamesBook = classOf[Book].getDeclaredFields.map(x => x.getName)
val columnNamesTransaction = clas-sOf[Transaction].getDeclaredFields.map(x => x.getName)

//Read data as Spark Dataframes
val book = spark.read.option("header", false).option("delimiter","|")
  .csv(hdfsFilePath  + "/books/books").toDF(columnNamesBook:_*).as[Book]
val transaction = spark.read.option("header", false).option("delimiter","|")
  .csv(hdfsFilePath  + "/transactions/transactions").toDF(columnNamesTransaction:_*).as[Transaction]

//Join book and transactions dataframes together and filter based on conditions
val data = book
  .withColumnRenamed("bk_publisher","book_publisher")
  .withColumnRenamed("bk_price","book_price")
  .join(transaction
    .withColumnRenamed("trans_id","transaction_id")
  )
  .where($"trans_id" === $"book_id")
  .filter(year($"trans_transaction_dt".cast(TimestampType)).between("2008", "2010"))
  .select($"book_publisher", $"book_price".cast(DoubleType), $"trans_quantity".cast(IntegerType))
  .groupBy("book_publisher")
  .agg(round(sum($"book_price" * $"trans_quantity"),2).alias("total_sales"))
  .orderBy(desc("total_sales"))
  .limit(10)
data.show()
```

5 RESULTS AND ANALYSIS

Both the MapReduce and Spark job has been subjected to the same amount of task relatively. The queries jobs have been in the order of low complexity (Scan Query) to the most complex (Three-Way Join). The scan query is more of scan through and filtering tasks for both MapReduce and Spark jobs. The aggregation combines scanning, grouping, or removing before generating results. The two-way join combines two tables which double the tasks and increase its complexity. The three-way join is the most complex of all the tasks obviously, and it tests the efficiency of both MapReduce and Spark job. The time taken for each of these tasks to be completed been monitored and recorded and will be analyzed.

5.1 Results for MapReduce and Spark Jobs

The following results were obtained for the time taken for both MapReduce and Spark jobs to be completed.

i. The MapReduce Scan Task

This task takes 31.40 seconds to be completed as shown in Figure 24.

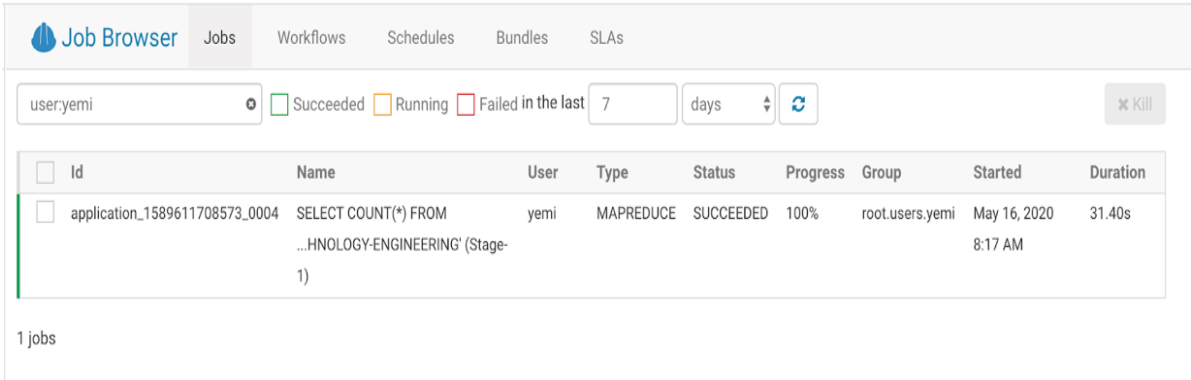

Figure 24. The MapReduce Scan Task Result.

ii. The Spark Scan Task

It takes 41 seconds for Spark to accomplish its scan query task as shown in Figure 25.

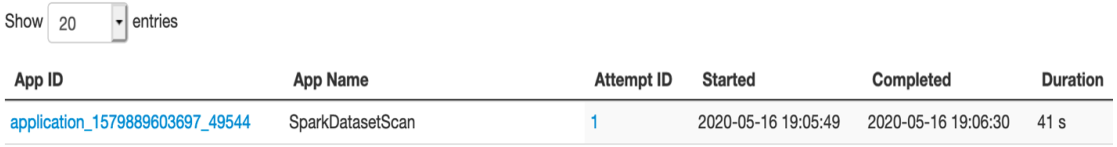

Figure 25. The Spark Scan Task Result

iii. The MapReduce Aggregation Task

It takes about 66s for MapReduce to complete its aggregation query task as shown below in Figure 26.

Figure 26. MapReduce Aggregation Task Result

iv. The Spark Aggregation Task

It takes about 35 seconds for Spark to perform its aggregation tasks as shown below in Figure 27.

Show 20 entries

App ID	App Name	Attempt ID	Started	Completed	Duration
application_1579889603697_49545	SparkAggregation	1	2020-05-16 19:17:19	2020-05-16 19:17:54	35 s

Figure 27. Spark Aggregation Task Result

v. The MapReduce Two-Way Join Task

The MapReduce takes about 175 seconds to complete this task as shown in Figure 28.

	(Stage-2)							
application_1589611708573_0020	SELECT tmp.book_category, ROUND(tmp.rev...10 (Stage-1)	yemi	MAPREDUCE	SUCCEEDED	100%	root.users.yemi	May 16, 2020 8:35 AM	2m, 55s

Figure 28. The MapReduce Two-Way Join Task Result

vi. The Spark Two-Way Join Task

It takes Spark 66 seconds to complete two-way join task as shown in Figure 28.

Show 20 entries

App ID	App Name	Attempt ID	Started	Completed	Duration
application_1579889603697_49546	SparkTwoWayJoin	1	2020-05-16 19:21:04	2020-05-16 19:22:11	1.1 min

Figure 29. Spark Two-Way Join Task Result

vii. The MapReduce Three-Way Join Task

It takes about 221 seconds for MapReduce to complete the most complex of its task as shown in Figure 30.

Figure 30. MapReduce Three-Way Join Task Results

viii. The Spark Three-Way Join Task

Sparks takes 78 seconds to complete the most complex of all its tasks as shown in Figure 31.

Figure 31. Spark Three-Way Join Task Results

5.2 Analysis

The Table 1 shows the recorded time in seconds it takes both MapReduce and Spark to complete their tasks. At the simplest task, which is Scan Query Job, the MapReduce was effective. It takes Spark longer time to complete its task compared to MapReduce. This is because of MapReduce being effective with batch data processing. This is due to

its linear data processing capabilities with large data. The simplest task being just scanning, and filtering might have given MapReduce the observed advantage.

As the tasks get tougher, it is observed that Spark complete its tasks faster as expected. It takes MapReduce approximately three times what Spark requires to complete its tasks. This is probably because of Spark in memory processing capabilities than that of MapReduce input/output disk latency which makes the later slower.

Scenarios (Query Jobs Performed)	MapReduce Job Time Elapsed (s)	Spark Job Time Elapsed (s)
Scenario 1 (Scan Query)	31.40	41
Scenario 2 (Aggregation Query)	66	35
Scenario 3 (Two Way Join Query)	175	66
Scenario 4 (Three Way Join Query)	221	78

Table 1. Time Elapsed for MapReduce and Spark Job

6 CONCLUSION

This thesis work compares the performances of both MapReduce and Apache Spark. We launched and configured four Linux Instances using AWS. We selected a t2.micro type instances from the Amazon EC2 console. The four Linux Instances represent four cluster nodes needed for benchmarking, one master and three slaves. The four Linux instances were connected to the terminal through SSH. The Linux Instances were granted access to each other from the master to the slaves. VPC was configured to enable our Linux Instances being launched into a pre-defined virtual network. A security group was associated with the launched Linux Instances with inbound and outbound rules specified.

We created a key pair that allows connection to the Linux Instances through SSH. The Linux Instances were launched into a public subnet and associated with an Elastic IP addresses. The Elastic IP ensures that the Linux Instances have permanent IPs whenever they are launched. The CDH was installed using Cloudera Manager 5.16 version. The four cluster nodes were installed immediately CDH was launched into the web. These services HDFS, Hue, Oozie, and YARN were added to the cluster. Service roles were added to the master and the slaves. Big Data files of 5 Gigabyte each for Books, Customers, and Transactions were created. We copied the data files into HDFS and a Hive table was created.

We started benchmarking the performances of MapReduce and Apache Spark by observing task execution time. We carried out the benchmark by running four different types of queries namely: Scan query, Aggregate query, Two-way join query and Three-way join query. These queries are in order of complexity with the Scan query being the easiest, while Three-way join is the most difficult. The time taken for each query to run for both MapReduce jobs and Spark jobs were tabulated. We critically observed the recorded time taken for each task completion.

In scenario 1, which is the Scan query it took 31.40s for MapReduce job to complete while Spark took 41s. We observed that MapReduce was efficient at the simplest task. In scenario 2, which is Aggregation job MapReduce jobs took 66s to complete while Spark took 35s. We observed that Spark performs better than MapReduce at this job. In scenario 3, we carried out Two-way join query. We observed that Spark performs even better at this stage more than the previous job. Sparks job took 66s to complete while MapReduce took 175s. In scenario 4, we carried out a Three-way join query which is the most difficult of all the tasks. We observed that MapReduce job took 221s to complete while Spark took only 78s. Spark performs even more better at this stage completing its job at almost 3 times faster than MapReduce.

We observed that at the simplest task, which was Scan query, MapReduce was effective. Its execution time was shorter compared to that of Spar. This demonstrates that MapReduce is efficient with linear data processing of big data. It enables parallel processing where more processor executes divided jobs. However, Apache Spark performed better as the tasks get tougher. In fact, at the most difficult job, which is Three-way join, Spark performs almost three times faster than MapReduce. This demonstrates the in-memory data processing ability of Spark. It saves the intermediate output in the memory which practically reduces execution time for all its tasks. It was concluded that Apache Spark is more effective and faster overall.

The challenges of this research work are that AWS is very expensive for a student experimenting with this without funding. It was practically difficult to use the free tier eligible Linux Instances due to a lot of limitations such as low memory sizes. For the future work, it will be interesting to measure the throughput of each cluster nodes for both MapReduce and Apache Spark jobs when both subjected to the same big data task.

REFERENCES

A. G. Prajapati, S. J. Sharma and V. S. Badgujar, "All About Cloud: A Systematic Survey," 2018 International Conference on Smart City and Emerging Technology (ICSCET), Mumbai, 2018, pp. 1-6, doi: 10.1109/ICSCET.2018.8537277. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8537277

A. K. Manekar and G. Pradeepini, "Cloud Based Big Data Analytics a Review," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Jabalpur, 2015, pp. 785-788. [online] Available at:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7546203.

Amazon Elastic MapReduce: Developer Guide 2014. [online] Available at: https://s3.amazonaws.com/awsdocs/ElasticMapReduce/latest/emr-dg.pdf

A. Verma, A. H. Mansuri and N. Jain, "Big data management processing with Hadoop MapReduce and spark technology: A comparison," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-4. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7570891

A. V. Hazarika, G. J. S. R. Ram and E. Jain, "Performance comparision of Hadoop and spark engine," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, 2017, pp. 671-674. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8058263

B. Akil, Y. Zhou and U. Röhm, "On the usability of Hadoop MapReduce, Apache Spark & Apache flink for data science," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 303-310. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8257938&tag=1

Cloud computing with AWS. [online] Available at: Available at https://aws.amazon.com/what-is-aws/?nc1=f_cc

Cloudera Enterprise 5.14.x CDH Overview. 2020. [online] Available at: https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/cdh_intro.html

Cloudera Manager Overview Cloudera Enterprise 6.3.x. 2020. [online] Available at: https://docs.cloudera.com/documentation/enterprise/latest/topics/cm_intro_primer.html

Getting started with Amazon EC2 Linux instances. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2ug.pdf#EC2_GetStarted

Hadoop Ecosystem and Their Components – A Complete Tutorial by DataFlair Team · Updated April 26, 2019. [online] Available at: https://data-flair.training/blogs/hadoop-ecosystem-components/

Hadoop Mapper–4 Steps Learning to MapReduce Mapper by DataFlair Team Updated November 21, 2018. [online] Available at: https://data-flair.training/blogs/hadoop-mapper-in-mapreduce/

Hadoop Reducer–3 Steps learning for MapReduce Reducer by DataFlair Team Updated November 21, 2018. [online] Available at: https://data-flair.training/blogs/hadoop-reducer/

J. Fu, J. Sun and K. Wang, "SPARK – A Big Data Processing Platform for Machine Learning," 2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), Wuhan, 2016, pp. 48-51,. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7823490

Kalbandi, Ishwarappa & Anuradha, J. (2015). A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. Procedia Computer Science. [online] Available at: https://www.researchgate.net/publication/282536587_A_

Brief_Introduction_on_Big_Data_5Vs_Characteristics_and_Hadoop_Technology/citation/download

K. Aziz, D. Zaidouni and M. Bellafkih, "Real-time data analysis using Spark and Hadoop," 2018 4th International Conference on Optimization and Applications (ICOA), Mohammedia, 2018, pp. 1-6. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8370593

KSpark RDD – Introduction, Features & Operations of RDD by DataFlair Team Updated May 7, 2019. [online] Available at: https://data-flair.training/blogs/spark-rdd-tutorial/

Learn the Concept of Key-Value Pair in Hadoop MapReduce by DataFlair Team · Updated · November 16, 2018. [online] Available at: https://data-flair.training/blogs/key-value-pair-in-hadoop-mapreduce/

M. A. Giraldo, J. F. Duitama and J. D. Arias-Londoño, "MapReduce and Spark-based architecture for bi-class classification using SVM," 2018 IEEE 1st Colombian Conference on Applications in Computational Intelligence (ColCACI), Medellin, 2018, pp. 1-6. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=& arnumber =8484855

M. A. Hedjazi, I. Kourbane, Y. Genc and B. Ali, "A comparison of Hadoop, Spark and Storm for the task of large scale image classification," 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, 2018, pp. 1-4. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber= 8404688

M. Khan, Salman and N. Iqbal, "Computational Performance Analysis of Cluster-based Technologies for Big Data Analytics," 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE

Smart Data (SmartData), Exeter, 2017, pp. 280-286 [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8276765

M. T. Khorshed, N. A. Sharma, A. V. Dutt, A. B. M. S. Ali and Y. Xiang, "Real time cyber attack analysis on Hadoop ecosystem using machine learning algorithms," 2015 2nd Asia-Pacific World Congress on Computer Science and Engineering [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7476223

M. Zhou, R. Zhang, D. Zeng and W. Qian, "Services in the Cloud Computing era: A survey," 2010 4th International Universal Communication Symposium, Beijing, 2010, pp. 40-46, doi: 10.1109/IUCS.2010.5666772. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5666772

NiketanPansare1, VinayakBorkar2, ChrisJermaine1, TysonCondie OnlineAggregation-forLarge MapReduceJobs. [online] Available at: https://asterix.ics.uci.edu/pub/vldb11-oa.pdf

R. K. Chawda and G. Thakur, "Big data and advanced analytics tools," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-8. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7570890.

Shengti Pan (2016) The Performance Comparison of Hadoop and Spark: CULMINATING PROJECTS IN COMPUTER SCIENCE [online] Available at: https://pdfs.semanticscholar.org/fab7/ef9a4d46d1af7ac5e26cf0146bf07decea05.pdf.ieee.org/stamp/stamp.jsp?tp=&arnumber=7570890.

Talia, Domenico. (2013). Clouds for Scalable Big Data Analytics. Computer. 46. 98-101. 10.1109/MC.2013.162. Extracting useful knowledge from huge digital datasets requires smart and scalable analytics services, programming tools, and applications. [online] Available at: https://www.researchgate.net/profile/Domenico

_Talia/publication/260584533_Clouds_for_Scalable_Big_Data_Analytics/links/5cff84a74585157d15a21995/Clouds-for-Scalable-Big-Data-Analytics.pdf

The Hadoop Ecosystem: Welcome to the zoo!. [online] Available at: https://www.coursera.org/lecture/big-data-introduction/the-hadoop-ecosystem-welcome-to-the-zoo-BpHNu

X. Lu, M. W. U. Rahman, N. Islam, D. Shankar and D. K. Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences," 2014 IEEE 22nd Annual Symposium on High-Performance Interconnects, Mountain View, CA, 2014 , pp. 9-16, doi: 10.1109/HOTI.2014.15. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6925713