Optimisation of a Hadoop Cluster Based on SDN in Cloud Computing for Big Data Applications

By

Ali Khaleel

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy

Department of Electronic and Computer Engineering
College of Engineering, Design and Physical Sciences
Brunel University London, United Kingdom

July 2018

Abstract

Big data has received a great deal attention from many sectors, including academia, industry and government. The Hadoop framework has emerged for supporting its storage and analysis using the MapReduce programming module. However, this framework is a complex system that has more than 150 parameters and some of them can exert a considerable effect on the performance of a Hadoop job. The optimum tuning of the Hadoop parameters is a difficult task as well as being time consuming. In this thesis, an optimisation approach is presented to improve the performance of a Hadoop framework by setting the values of the Hadoop parameters automatically. Specifically, genetic programming is used to construct a fitness function that represents the interrelations among the Hadoop parameters. Then, a genetic algorithm is employed to search for the optimum or near the optimum values of the Hadoop parameters. A Hadoop cluster is configured on two severe at Brunel University London to evaluate the performance of the proposed optimisation approach. The experimental results show that the performance of a Hadoop MapReduce job for 20 GB on Word Count Application is improved by 69.63% and 30.31% when compared to the default settings and state of the art, respectively. Whilst on Tera sort application, it is improved by 73.39% and 55.93%. For better optimisation, SDN is also employed to improve the performance of a Hadoop job. The experimental results show that the performance of a Hadoop job in SDN network for 50 GB is improved by 32.8% when compared to traditional network. Whilst on Tera sort application, the improvement for 50 GB is on average 38.7%. An effective computing platform is also presented in this thesis to support solar irradiation data analytics. It is built based on RHIPE to provide fast analysis and calculation for solar irradiation datasets. The performance of RHIPE is compared with the R language in terms of accuracy, scalability and speedup. The speed up of RHIPE is evaluated by Gustafson's Law, which is revised to enhance the performance of the parallel computation on intensive irradiation data sets in a cluster computing environment like Hadoop. The performance of the proposed work is evaluated using a Hadoop cluster based on the Microsoft azure cloud and the experimental results show that RHIPE provides considerable improvements over the R language. Finally, an effective routing algorithm based on SDN to improve the performance of a Hadoop job in a large scale cluster in a data centre network is presented. The proposed algorithm is used to improve the performance of a Hadoop job during the shuffle phase by allocating efficient paths for each shuffling flow, according to the network resources demand of each flow as well as their size and number. Furthermore, it is also employed to allocate alternative paths for each shuffling flow in the case of any link crashing or failure. This algorithm is evaluated by two network topologies, namely, fat tree and leaf-spine, built by EstiNet emulator software. The experimental results show that the proposed approach improves the performance of a Hadoop job in a data centre network.

Acknowledgments

Firstly, I would like to express my gratefulness and appreciation to the Ministry of Higher Education and Scientific Research (MOHESR), Iraqi Cultural Attaché and University of Diyala in Iraq for funding and supporting this study.

I also would like to express my sincere thankfulness to Professor Hamed Al-Raweshidy for his useful advices, guidance, and encouragement through the years. His unstopping belief, enthusiastic support and beneficial discussions with him are very precious and helpful. Most appreciated thing is his understanding and patience while allowing me enough room to choose the study topics on my own.

Also, I would like to thank my family for their moral support, love, inspiration and encouragement during the years of my study. They helped me to pass through the most difficult time of my life.

List of Publications

Accepted papers

[1] Ali Khaleel, Hamed Al-Raweshidy ’’ Solar Radiation Time Series Analytics Based on Hadoop Integrated Programming Language Environment (RHIPE)’’ in *IEEE SAI Intelligent Systems Conference*, London, UK, 2018.

Published papers

[2] Ali Khaleel, Hamed Al-Raweshidy ” Optimisation of Hadoop MapReduce Configuration Parameter Settings using Genetic Algorithms’’ in *IEEE SAI Computing Conference*, London, UK, 2018, DOI: 10.1007/978-3-030-01177-2_4}.

[3] Ali Khaleel, Hamed Saffa Al-Raweshidy ’’ Effective Routing algorithm based on Software defined networking for big data applications in Data centre network’’ in the *16th IEEE International Conference on Pervasive Intelligence and Computing*, Athens, Greece, 2018, DOI**:** 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00-20.

[4] Ali Khaleel, Hamed Al-Raweshidy ’’Optimisation of computing and networking resources of a Hadoop cluster based on software defined network’’ in IEEE access journal, DOI: 10.1109/ACCESS.2018.2876385.

Table of Contents

List of Figures

List of Tables

Abbreviations

APIs	Application program interfaces
BGP	Border gateway protocol
CapEx	Capital expenditure
CPU	Central processing unit
DCN	Data centre network
EC2	Elastic cloud computing
ECMP	Equal cost multipath algorithm
EMR	Elastic MapReduce
GA	Genetic Algorithm
GFS	Google File System
GP	Genetic Programming
HA	High availability
HBase	Hadoop database
HDFS	Hadoop distributed file system
HPC	High performance computing
IoT	Internet of things
IT	Information technology
LLDP	link layer discovery protocol
MPI	Message passing interface programming
OPEX	Operational expenditure
OSPF	Open shortest path first
Open vSwitch	Open virtual switch
PVM	Parallel virtual machine

PC	Personal computer
RAM	Random-access memory
RDBMS	Relational database management systems
RHIPE	R and Hadoop Integrated Programming Environment
RISC	Reduced instruction set computer
SDN	Software defined networking
SPB	Shortest Path Bridging
SPOF	Single-Point-of-Failure
SMP	Symmetric multi-processing
TCP	Transmission Control Protocol
TRILL	Transparent Interconnection of Lots of Links
VM	Virtual machine
VRRP	Virtual router redundancy protocol
VLAN	Virtual Local Area Network

Chapter 1

Introduction

The emergence of smartphones, internet of things and social media has a significant impact on the data generation. Different types of data sets are generating from various sources including the wide use of YouTube, google and social networking websites like Facebook and twitter. Numerous data sets are also generated from other sources, such as online mobile banking, online retail sector services, healthcare system, education systems, sensors and other online services. Many companies and organisations also generate huge amounts of data every day. Around 2.3 Zettabytes of data sets are generated every day around the world. This number of generated data will increase in 2020 to be 40 zettabytes. 100 terabytes are stored by many companies only in the United States [1]. These massive amounts of various data sets are classified as a big data. The term of big data refers to the large scale of data sets that are difficult to be stored, analysed and processed by the conventional tools due to their diversity and complexity. Big data can be included into the form of structured model where the data is organised and fit into the conventional databases and can be easily analysed and processed using the traditional tools. Another form of the big data is the unstructured model, which represents the most challenging part, because the large data in this case is produced from different sources with different types of data including images, texts, videos, audio files and web pages. This type of data is not organised in pre-defined format and thus, it cannot be fit into the conventional databases or be processed by the traditional tools [2].

Big data can be characterised by 5vs [3], which are volume, variety, velocity, value and veracity. The volume refers to the massive data sets, which are generated by many users, companies and organisations around the world consisting of zettabytes. On the other hand, the variety is another complex characteristic where massive data sets are generated with different formats including both of structured and unstructured data, such as videos, images, texts, audios and data logs. The third characteristic of big data is the velocity where the data moves at fast pace form point to another. The velocity also refers to the frequently of the generated data every seconds, minute, month and year. The generated data can be processed in real time or in batch mode when only

required. The veracity in big data refers to the large number of uncertain and imprecise data that should be sorted to achieve clean and authentic data. One of the significant aspects is to obtain meaningful value of big data. Many companies have started to analyse their own data to improve their own products. In the healthcare system, analysing the data might lead to figure out some diseases and even find appropriate remedies for them.

As we mentioned, big data can be generated from various sources due to the rapid advance in the computer and communication networks. Today many people around the world use smart phones and generate large data by browsing many websites and social groups using the internet. The number of connected devices to the internet will reach to 50 billion devices in 2020. Machine to machine communication (M2M) [4] also contributes in producing large number of data sets without any human intervention where sensor devices are communicated with each other to exchange information. Many sensors are used by many people and companies to achieve intelligent decisions using internet of things (IoT) technology [5]. According to Gartner [6], around 6.4 billion things and devices were connected to the internet in 2016 and about 8.4 billion in 2017. This number is expected to increase to more than 20 billion in 2020. Many companies and governments need to analyse the huge data sets that come into different types and formats to extract useful information in order to improve their own businesses. However, processing and storing such massive data has become a challenging task for the existing traditional tools like relational database management systems (RDBMS), because it lacks the scalability of large data sets and it supports only the structured data that has pre-defined format. However, a solution has been reached to store and analyse these large data sets in effective way [7]. The solution can be used to provide distributed computing for parallel processing and storage by applying Hadoop MapReduce framework.

The framework of Hadoop and its programming technique MapReduce can be implemented across a cluster of computing nodes. The cluster can be run in cloud computing environment such as Amazon Elastic Compute Cloud (Amazon EC2) and Microsoft Azure. Cloud computing is the technology that enables customers to access and share the computing resources using the internet. The computing resources in the cloud computing consists of networks, servers, storage, applications, services and software that can be accessed by users when they require them. Companies and users

can access information anytime and anywhere from any geographical point in the world whenever the internet is available. The computing resources can be utilised by multiple customers using remote access. Furthermore, Cloud computing provides the opportunity for individuals and businesses to use the software and hardware that are managed by third parties remotely [8] [9]. They provide the access to share and dynamically allocate/de-allocate the computing resources depends on user demand. Amazon Elastic Compute Cloud (Amazon EC2) provides virtual computing resources to be accessed by users based on pay as you go to run their own applications on allocated resources. Virtual computing resources can be provided as instances [10] [11].

More details about cloud computing are provided in chapter 2. MapReduce programming model can be used to support big data intensive applications in cloud computing environment. MapReduce is a programming technique model that supports parallel processing and analysis for massive unstructured datasets [12]. Amazon provides elastic MapReduce (EMR) for big data processing using elastic cloud computing (EC2) machines. MapReduce can also be implemented using indoor-cluster consisting of several machines by installing Apache Hadoop on each node. Hadoop framework [13] is a distributed computing platform that is built in java and released by the Apache foundation to enable the implementation of MapReduce programming technique on huge data sets. Apache Hadoop has been widely used by researchers and companies to analyse, process and store massive amounts of unstructured data in distributed environment. Hadoop is characterised by some salient features such as scalability, resiliently, fault tolerance and the parallelisation nature [14]. Both the MapReduce model and Hadoop framework are explained in details in chapter 2. The Next section presents the aim and objectives of the thesis.

1.1 Aims and Objectives

The major aims and objectives of this thesis were as follows:

- Aims:

Reduce the execution time of a Hadoop job and accelerate the processing time of large datasets in Hadoop framework.

- Objectives:

1- Improve the performance of a Hadoop framework by applying genetic algorithms to better utilising the computing resources in terms of CPU, Memory and hard disk I/O.

2- Integrate R language with Hadoop framework to enable R expressions to be executed across multiple computing nodes in a cluster computing environment, and evaluate the performance of both R and Hadoop from different aspects including scalability, accuracy and speedup.

3- Improve the networking part of a Hadoop framework in large scale clusters to speed up the processing time of a Hadoop job during the shuffle phase.

1.2 Motivations

This thesis has number of motivations

- Hadoop MapReduce framework is the main computing platform used by many businesses to support the analysis and the processing of big data. However, the parameters of Hadoop framework can influence the processing time of a Hadoop MapReduce job and hence, affect the overall performance. According to recent studies, the tuning of Hadoop parameters with the right values can have a considerable effect on the Hadoop MapReduce performance. Hadoop has more than 150 parameters that manage the computing resources specified for the Hadoop MapReduce job. The execution of Hadoop MapReduce jobs are performed using the default values of Hadoop configuration parameters. However, the performance of Hadoop MapReduce job can be affected negatively, as a result of the ineffective utilisation of the computing resources using the default values of the parameters. Therefore, tuning the configurations of the Hadoop parameters with optimal or near optimal values can have a positive influence on the overall performance of the Hadoop MapReduce job. However, because of the large number and the complexity of the Hadoop parameters, manual tuning has become a challenging task. As a consequence, dynamic and influential technique is necessarily required to tune these parameters.

- Solar energy has become one of the main sustainable and renewable energy sources. It provides many features to the world, such as bill cost reduction, emission reduction,

green energy and low maintenance. However, one of the main issues of the solar energy is the unpredictability of the solar irradiation that mainly depends on the weather conditions. It is very important to predict and calculate the output power of the solar arrays based on the solar irradiation. Calculation of the large number of solar irradiation data is a challenging task using the traditional tools of time series analysis like R language. As a result, a novel method based on a Hadoop framework is highly needed to utilise the computing resources to achieve fast calculation and timely analysis of the solar irradiation.

- MapReduce programming model has been widely deployed for big data analytic. It can be implemented based on Apache Hadoop using several computing nodes in a cluster computing environment. The computing resources in a Hadoop framework, such as memory RAM and hard disk I/O can be scaled by using several servers to store and process massive amounts of data sets. However, increase number of servers would result in increasing the network traffic of the entire network. The communication and network aspect of the Hadoop MapReduce framework can affect the overall processing time of a MapReduce job especially when the communication patterns of MapReduce are heavy. The reason is that the networking aspect for job scheduling is not considered by the default Hadoop resource manager. During the parallel and distributed processing of large data sets in Hadoop cluster using MapReduce, many flows are exchanged between hosts especially during the shuffling phase. Each flow should be allocated a particular amount of network bandwidth for rapid and effective processing. However, the traditional legacy network is not able to dynamically allocate efficient bandwidth for each flow. Hence, dynamic scheduling and routing algorithm for big data applications using Hadoop MapReduce based on software defined networking (SDN) has become a necessity.

1.3 Research Methodology

In this research, the performance of a Hadoop MapReduce job was evaluated by applying both genetic programming and genetic algorithm to optimise Hadoop MapReduce parameters by tuning them automatically. Software defined networking is

also employed on Hadoop cluster to evaluate the performance of a Hadoop MapReduce job with the optimised values of Hadoop parameters compared to the performance in conventional networks. Another work was performed on the networking side of a Hadoop cluster to improve the performance of the Hadoop MapReduce job using software defined network in small and large scale clusters.

The proposed work was evaluated using a Hadoop cluster with two servers (Intel Xeon X5550 server 1 and uxisvm04 server 2). Different sizes of virtual machines ranging from 8-14 VMs have been created using the oracle virtual box. Hadoop Cloudera (Hadoop 2.6.0-cdh5.9.0) has been installed on each virtual machine (VM). All virtual machines were placed on Microsoft azure cloud. We assigned for each VM 8 GB of RAM and 4 CPU cores. Total storage of 320 GB was allocated for the cluster. The Ubuntu 14.04 TLS operating system was installed on each VM and X2G client was used as a graphical user interface to access Hadoop cluster remotely. EstiNet emulator software was employed to build both the fat tree and leaf-spine topology of the data Centre network. SDN Floodlight controller was installed on one PC. SDN application and Open vSwitches were also used in the data centre network. More details about the cluster specifications of each experiment are available in chapters 3, 4&5.

1.4 Main Contributions

The main contributions of this thesis are summarised below:

1- An Optimisation technique is proposed in chapter 3 to optimise the parameter settings of Hadoop by automatically tuning them using both the genetic programming and genetic algorithm. Genetic programming is used to construct a fitness function based on the input samples datasets. The interrelations and reliance of the Hadoop parameters are represented by the fitness function. The optimisation technique of the Hadoop parameters uses the fitness function as an objective function. Genetic algorithm is used to search for the optimal values of the Hadoop parameters. Software defined networking is employed in a Hadoop cluster to evaluate the performance of a Hadoop MapReduce job using the optimised values that identified by the genetic algorithm compared to conventional network.

2- Chapter 4 presents a distributed, scalable, fault tolerant and effective computing platform to store and analyse large solar irradiation data sets. The proposed platform uses R and Hadoop Integrated Programming Environment (RHIPE) for fast analysis and statistical calculations on massive irradiation data sets. RHIPE is evaluated in terms of scalability, accuracy and speedup compared with R language. Gustafson's Law is used to analyse the speed up of parallel RHIPE in a Hadoop cluster. An amendment on Gustafson's Law is made to improve the performance of the parallel computation on intensive data sets in a Hadoop cluster based on cloud computing environment.

3- In chapter 5, software defined networking is proposed to improve the performance of a Hadoop MapReduce job in a data centre network by providing network aware application. An Effective routing algorithm is implemented based on software defined networking to speed up the execution time of a Hadoop MapReduce job by dynamically improving the networking aspect during the shuffling phase. Both the fat tree topology and leaf-spine topology are used for the data centre network, so as to evaluate the performance of a Hadoop MapReduce job using the proposed work. The generated traffic from the shuffling phase of a MapReduce job in a Hadoop cluster in a data centre network is allocated with sufficient amount of network bandwidth using the proposed routing algorithm based on a software defined network.

4- The performance of the proposed work has been evaluated using a Hadoop cluster placed on internal cloud managed by Microsoft azure in Brunel University London. SDN Floodlight controller, SDN applications and Open vSwitch packages have been installed separately on physical nodes for further evaluations and investigations.

1.5 Thesis Organisation

The rest of this thesis is organised as follows:

In chapter 2, general background on the MapReduce programming technique and the Hadoop framework including the Hadoop distributed file system (HDFS) is provided. It also provides general overview on cloud computing, software defined network (SDN), OpenFlow and data centre network (DCN). The optimisation techniques of Hadoop MapReduce parameters are also introduced in this chapter. It provides the applications

of machine learning algorithm in Hadoop environment. Finally, this chapter provides some of the routing algorithms and network topologies used in the data centre network.

Chapter 3 presents an optimisation technique using both the genetic programming and Genetic algorithm to optimise Hadoop parameters. It first applies the genetic programming to construct a fitness function based on sample data sets. The genetic algorithm is applied to the constructed fitness function to find the optimum or near optimum values of the Hadoop parameters. A comparison is conducted between the performance of the proposed technique and the performance of Gunther and default settings. SDN is also employed to improve the performance of a Hadoop job further.

Chapter 4 presents a scalable and distributed framework for solar irradiation data storage and analysis. It uses R and Hadoop Integrated Programming Environment (RHIPE) for fast statistical calculation and analysis. The speedup of RHIPE is analysed using Gustafson's Law. The performance of RHIPE in terms of scalability, accuracy and speedup is compared with R language.

In chapter 5, an effective routing algorithm based on SDN for Hadoop MapReduce applications in a data centre network is presented. Both the fat tree topology and leaf-spine topology are used for the data centre network to evaluate the performance work. The proposed algorithm is compared with ECMP and TRILL to evaluate the performance of a Hadoop job in SDN network. The SDN network is also compared with the conventional one to evaluate the routing convergence time in both environments.

Chapter 6 concludes the thesis and discusses some research challenges. Some findings of the research are also discussed. Some considerations and proposals in future are also presented in this chapter.

Chapter2

Background

Hadoop is an open source framework that supports the parallel processing and distributed storage for massive data sets using the MapReduce programming model and the Hadoop distributed file system (HDFS). It has become the most popular framework for big data applications because of its remarkable characteristics such as scalability, flexibility, cost effectivity, and fault tolerance. It can be scaled up from a single server to thousands of servers in large clusters and data centre networks based on cloud computing environment. However, with the scaling of servers and machines for Hadoop in large scale clusters and data centre network, high volume of traffic will be generated. The emergence of software defined network brings the benefit of intelligent network management that can alleviate the traffic of big data applications in a Hadoop cluster. This chapter provides the architecture of Hadoop and its programing model MapReduce. It also provides an overview of software defined network and open flow protocol. Furthermore, it provides an overview of cloud computing and data centre network and presents some routing algorithms and network topologies used in the data centre network.

2.1 Hadoop Framework

Hadoop [15] is a distributed framework that is written in java, developed by Doug Cutting and Mike Cafarella, and was released as an open source by yahoo in 2008. The idea of Hadoop is inspired from the google file system (GFS) and google MapReduce. It enables the distributed processing for both structured and unstructured data by running MapReduce programming model using a cluster of servers and machines. With the drastic increase of data, many organisations are using Hadoop such as Facebook, twitter, yahoo, google and YouTube [16] due to its distributed nature and its ability to process and analyse enormous data sets. The most significant characteristics of a Hadoop framework are the scalability and fault tolerance. Hadoop can be implemented on large scale clusters with thousands of machines to bring the computation power and provide the distributed storage for massive data sets. The size of a Hadoop cluster

depends on the size of data sets and it can be scaled up with the increasing of data workload. The fault tolerance feature of the Hadoop framework provides the solution for any hardware or software failure in the cluster.

2.1.1 The Architecture of Hadoop Framework

The core components of the Hadoop framework consists of MapReduce and Hadoop distributed file system (HDFS). Figure 2.1 shows the architecture of Hadoop framework.

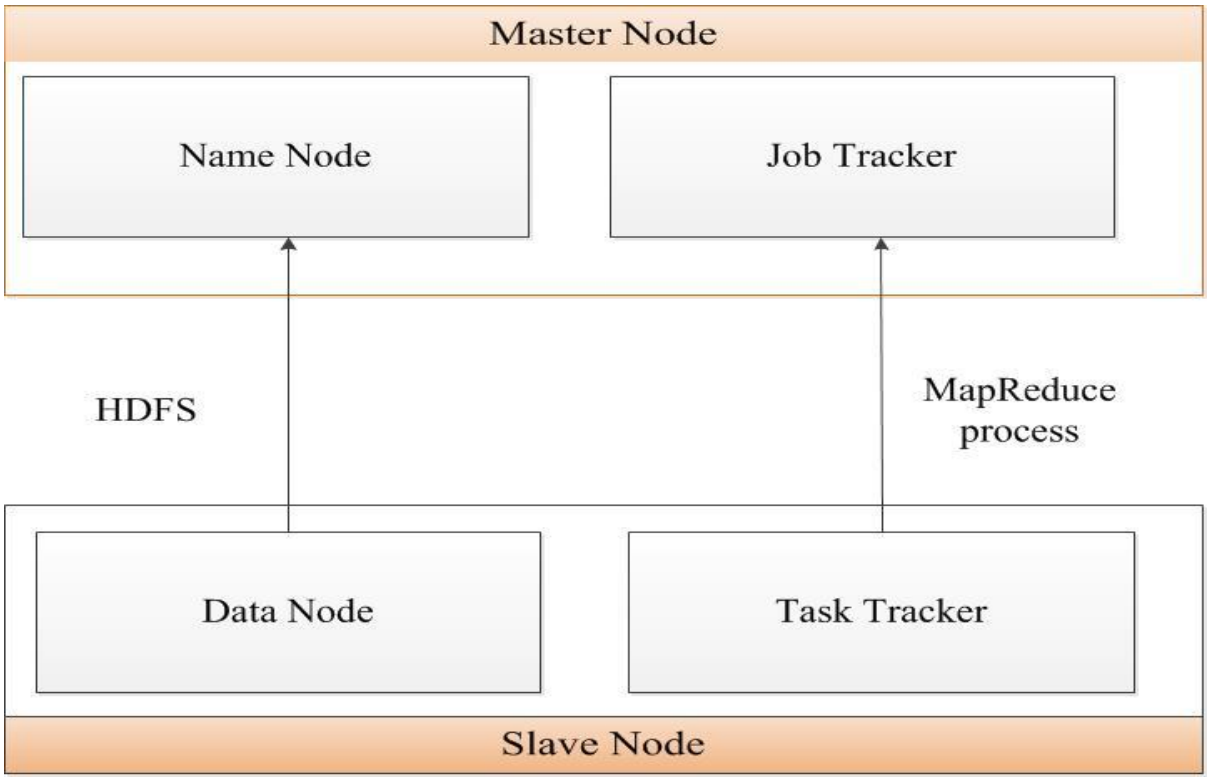

Figure 2. 1: Hadoop MapReduce Architecture

Hadoop has master/slave architecture, which includes one master node and several slave nodes. The master node contains one job tracker that monitors and manages the progress of MapReduce tasks, and assigning them to the task trackers. It also handles the failure and error cases of Hadoop nodes. The slave nodes contain the task trackers that interact with the job tracker in the master node. The task trackers are responsible to

execute all map and reduce tasks that run on the slave nodes (Data Nodes) by receiving instructions from the job tracker and they also process the data movement between the map phase and reduce phase. The job tracker receives heartbeat messages from each task tracker to monitor and update the progress of all running tasks in the cluster. However, if the job tracker does not receive any heartbeat messages from a certain task tracker, it reports that there is a failure occurred in the cluster node and coordinates the processing of this failure to any other working node in the cluster. Hence, Hadoop is the main powerful and popular platform for MapReduce programming model. The next section gives details of the MapReduce model.

2.2 MapReduce

MapReduce is a programming technique that is used to process and analyse massive amount of data sets on parallel in a distributed environment across a cluster of computing nodes [17]. It is the core component of Apache Hadoop that proposed by Jeffery Dean and Sanjay Ghemawat at Google in 2004. It performs the processing and computation of large data using map and reduce functions. The map and the reduce functions are executed in two different phases. The map function is executed at the map phase, whilst the reduce function is executed at the reduce phase. In the map function, the input data is divided into form of (key/value pairs) and produces the intermediate results of map phase in a list of (key/value pairs). All intermediate values are sorted and collected with the same associated key by the MapReduce library to pass them to the reducer in the reduce phase. Once the reducer receives the values associated with same key, it combines them together to produce the final output file.

Map function <k1, v1> $\longrightarrow$ list (<K2, v2>)

Reduce function <K2, list (v2)> $\longrightarrow$ list (<k3, v3>)

Figure 2.2 shows the process of input data sets using the MapReduce programming model. Different letters are presented as input data sets. We execute MapReduce algorithm to determine number of occurrences for each letter.

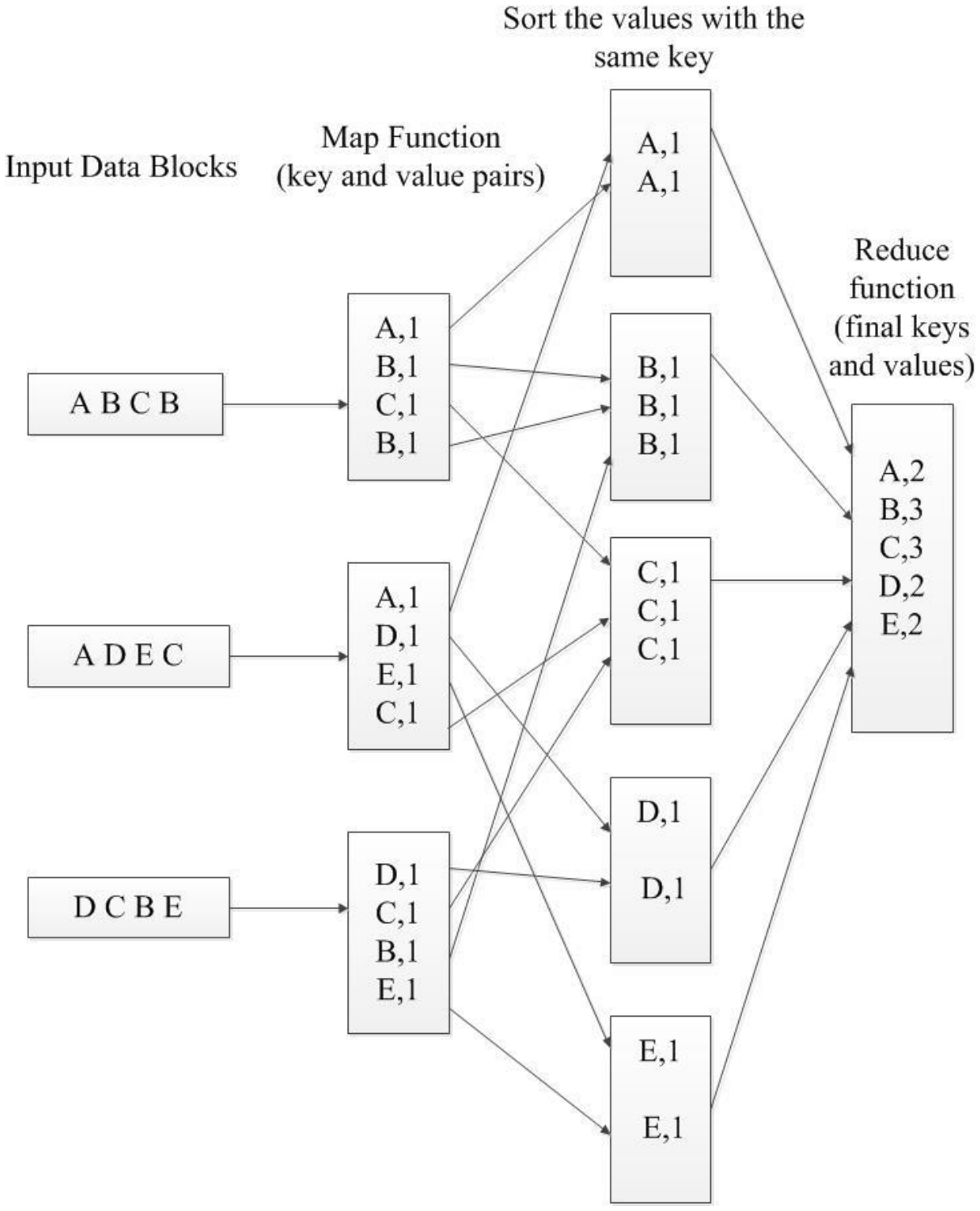

Figure 2. 2: The diagram of data process with MapReduce

MapReduce model can be applied in different fields such as health care applications, social networks, artificial intelligence, machine learning and web data processing. The

first use of MapReduce was in 2004 by google to construct its web index and it is also used by yahoo and Facebook. The processing of massive amounts of data sets is a challenging task and time consuming. As a result, MapReduce can be used due to its parallel and distributed nature that can distribute large data sets to be processed using a large number of commodity nodes. The other remarkable characteristics, such as fault tolerance, simplicity and high scalability can bring the effectiveness for processing and storing large heterogeneous data sets. MapReduce is highly scalable as it can easily scale the processing of massive data over multiple computing nodes. It partitions the input data into several blocks and schedules them to be executed by multiple cluster nodes and manage the communication between nodes in the cluster. It also can automatically handle any crashing or failure in the cluster without any effect on the computation process and the execution time. It supports parallelism in the cluster for processing intensive tasks and provides efficient implementation for different applications using map and reduce functions [18].

MapReduce is the main programming model of Apache Hadoop project [15]. Other projects are also implemented based on it. It is considered as the processing model of a Hadoop framework. The execution of a MapReduce job is performed in two stages. The first one is the map stage, where map job processes the input data sets that stored in the Hadoop distributed file system (HDFS). The mappers take the input data from the HDFS and divide it into several blocks. The size of data blocks can be configured with different size like 64MB or 128MB. All data blocks are distributed over different computing nodes in the cluster. Worker node reads the contents of each data block by assigning a map task for each block and apply user defined function. The intermediate results of map function are stored in a buffer memory. The results stored in the buffer memory are periodically written as partitions into local disk, which belongs to the cluster node that runs the map task. The locations of buffered results written on the local disk are moved back to the master node, which forwards them to the cluster node that runs the reduce task. In the second stage (also called the reduce stage), the reducers reads the results remotely from the local disks of mappers. The intermediate results are sorted by the reducers as groups of the same keys with their corresponding values. Furthermore, external sort is applied if the memory does not accommodate the intermediate output results. The reducers iterate over the sorted intermediate output results and move each unique key with its corresponding intermediate values to the

reduce function. The output results of reduce function in the reduce stage are written as *N* output files according to the number of reduce tasks. The final output files are stored into Hadoop distributed file system. Figure 2.3 shows the execution of MapReduce job.

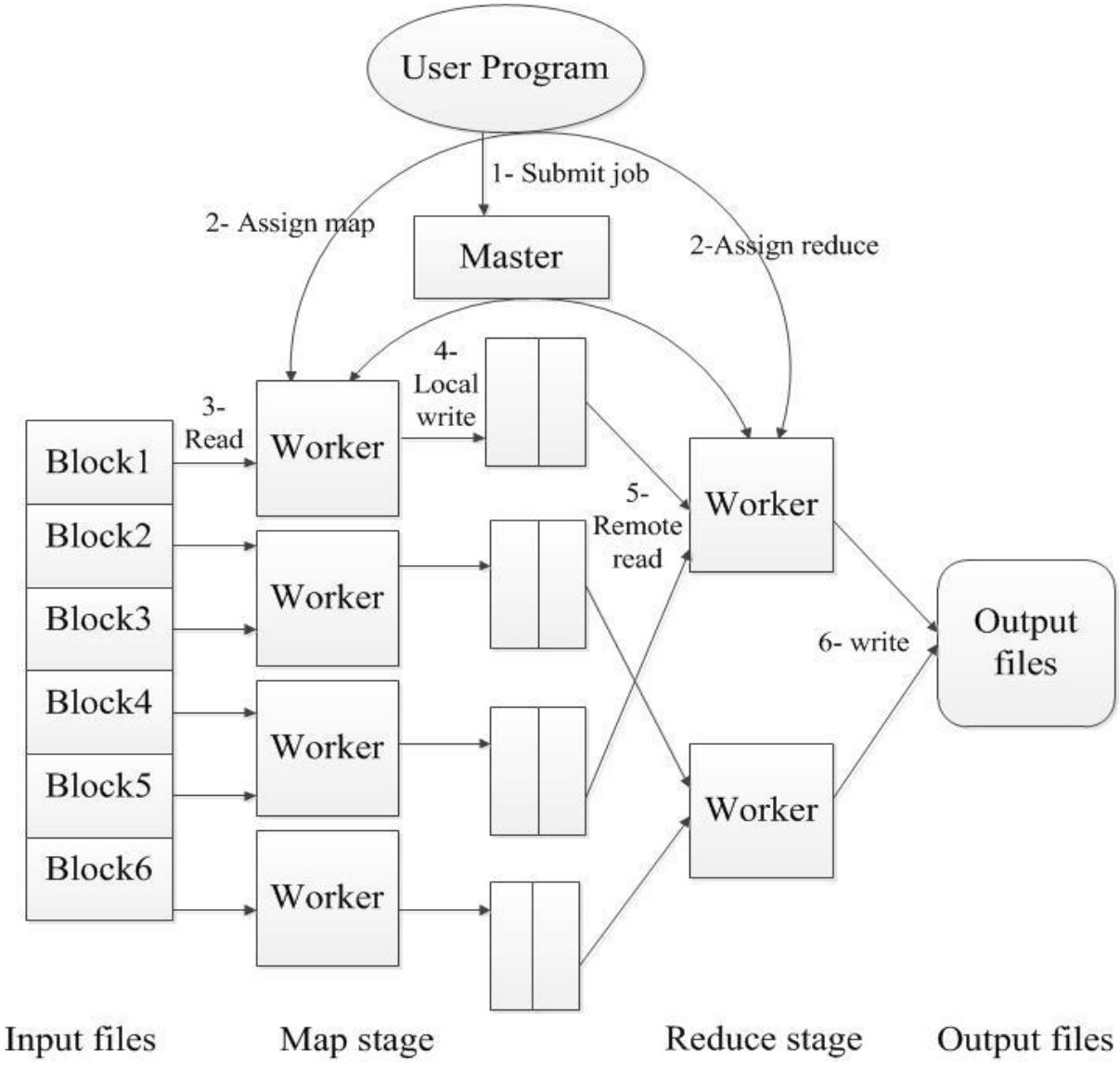

Figure 2. 3: MapReduce execution job

Map and reduce tasks are automatically distributed and executed in parallel by the MapReduce programming model across multiple cluster nodes or multiple processers in a single cluster node. The MapReduce model executes the map and reduce task simultaneously on a computing node in the cluster by specifying a number of map and reduce slots. Number of map and reduce slots depends on the amount of hardware

resources and the workload of Hadoop MapReduce framework. Map and reduce slots number should fit the characteristics and resources of the hardware in the cluster. Map and reduce slots are configured on the task tracker nodes through the configuration file. We suppose that there is 16 task tracker nodes (data nodes) and one map and reduce slot is assigned for each map and reduce task in the task tracker node, the total number of map and reduce slots is 16 slots. This number can be increased depends on the hardware resources and Hadoop workload to be 2 map and 2 reduce slots for each task in the task tracker node. However, the use of less map and reduce slots number than the map and reduce tasks leads to prolong the execution time.

2.3 Hadoop Distributed File System

With the continuous increase of datasets that are generated from different sources, it became a challenging task for the traditional tools to store and processes such massive data in a single device [19]. As a consequence, the partition and distribution of these massive datasets through a cluster of computer devices or several servers has become a vital solution. The datasets stored in different nodes of the cluster network are managed by a distributed file system. In Hadoop framework, the distributed file system is called HDFS (Hadoop distributed file system).

HDFS is responsible to store large scale datasets over several computers in the cluster network [20] [21]. The idea of HDFS is inspired from the Google File System (GFS) [22] [23] to run the applications of Hadoop MapReduce. It is a file system built in java and can provide reliable services for different users with different requirements and scalable storage for large datasets based on a considerable number of cluster nodes and commodity servers [24]. The key features of the HDFS, such as fault tolerance, scalability, reliability and availability have provided high streaming access to data applications. It is designed to execute a series of jobs in a batching processing mode. The data stored in the HDFS is divided into blocks and create multiple copies across different data nodes in the cluster. The size of each data block and the replication factor can be set in the configuration settings of a Hadoop cluster by a Hadoop user. Each data block is assigned with the same size and by default it is set to be 64MB and the replication number is 3. The replication of each data block is stored in different data nodes to support the fault tolerance feature and data availability in case of any failure or

crashing in one of the data nodes in the Hadoop cluster. HDFS is designed to support client/server architecture, where there is one server and several clients.

The architecture of HDFS system consists of two main components. The first one is the name node, which is responsible to control the file system namespace and manage the creation of files and directories. It also manages and regulates the client access to files and performs some file operations such as opening, closing and renaming files. The name node is a GNU/Linux operating system and software that run on the master node of Hadoop cluster and work as a server. Secondary name node is employed as a checkpoint node to assist the name node to achieve better performance. The secondary name node is not an alternative node or backup of the name node. The second component includes several data nodes that run on the slave nodes in the Hadoop cluster and work as clients that receive the instructions from the server (Name Node). One data node can be run on each slave or worker node. All data files stored in the slave nodes are managed by the data nodes. The data nodes also manage and control the creation, deletion and replication of data blocks in the cluster by receiving the instruction from the name node. Furthermore, they manage read/write operations on the file systems. The architecture of HDFS system is shown in Figure 2.4.

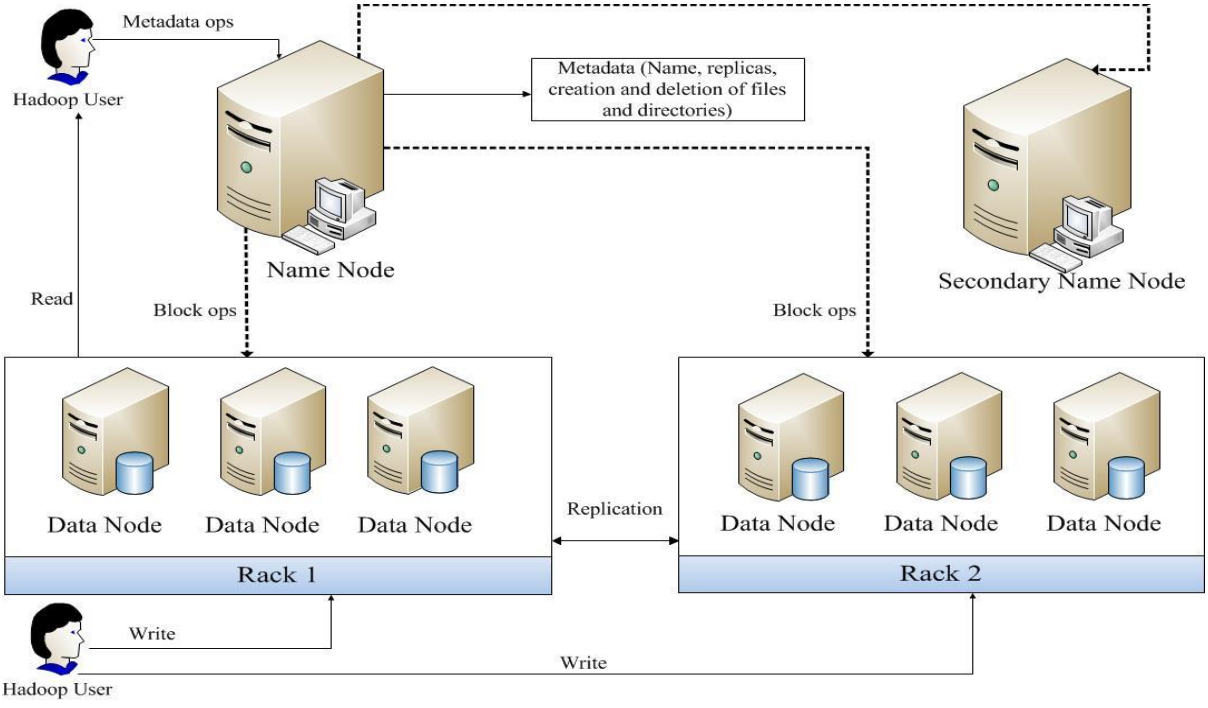

Figure 2. 4: The architecture of HDFS system

As can be seen in Figure 2.4, the metadata files that include the file system namespace, block ID, replication factor, replicas number for each file, the mapping process of blocks to data nodes, block replicas location on the data nodes are stored and maintained in the name node. On the other hand, the data nodes in the cluster manage the storage attached to them and store the multiple replicas of data blocks. These replicas are stored on different data nodes in the same rack or can be stored in different nodes in a different rack to obtain better performance of the cluster and increase the availability of data blocks.

2.4 Optimisation Techniques of Hadoop MapReduce Computing Resources

Massive data sets are generating every day from various sources and different fields including social media, financial sector, healthcare system, education and many more. As a result, this large number of generated data should be processed, analysed and stored on large number of connected devices in cluster computing environment like Hadoop framework. The huge amount of data sets is stressing the computing resources of the Hadoop cluster in terms of memory, CPU and hard disk I/O. Many optimisation techniques have been proposed to improve the performance of the Hadoop MapReduce framework. Some approaches have focused on the data locality improvement. Some other techniques have been developed to enhance the Job/Task scheduling of Hadoop MapReduce. Moreover, many techniques and algorithms have been proposed and implemented to optimise the parameter settings of the Hadoop framework to achieve better performance of a Hadoop MapReduce job. Some optimisation techniques of Hadoop parameters are presented in the following section.

2.4.1 The Optimisation Techniques of Hadoop MapReduce Parameters

Hadoop has become the most popular framework due to its salient features that enable the scalability, flexibility and the parallel processing in a distributed environment. However, it is a complex system, which contains a large number of complicated parts that communicate together through a number of cluster nodes. The processing time of jobs in a Hadoop framework can be affected by these parts as well as the hardware

resources, networking resources and the parameter settings of the Hadoop framework. Hadoop framework contains around 200 parameters and some of them can play a vital role in the processing time of Hadoop MapReduce jobs if they are configured properly with the optimum values. However, because of the complicated and the large number of these parameters, the optimal settings are becoming a challenging process as well as being time consuming. Therefore, considerable approaches and algorithms have been developed to adjust the parameter setting of the Hadoop system automatically. The proposed work in [25] have tuned the parameter settings of a Hadoop framework by applying Derivative-free optimization (DFO) technique to achieve optimised performance for applications. DFO uses DevOps tools to obtain the effective configurations of the cluster by applying the adjusted parameters according to the DFO decisions. In [26], an offline method was used to optimise the parameter configurations of the Hadoop framework. Multi- objective steady-state Non-dominated Sorting Genetic Algorithm II (ssNSGA-II) was employed to optimise two objectives, which are the resources utilisation of the instance and improve the execution time of Hadoop jobs. The optimum configuration settings are selected based on the two objectives. The proposed system has considered the dynamic machine instance type, but it ignored the dynamic cluster size.

Another work to optimise Hadoop parameters was conducted by [27] based on an adaptive recommendation system called mrEtalon that selects the near optimum parameter settings for the incoming job. The recommendation system uses a warehouse of configuration parameters to obtain the optimal one rapidly based on filtering method. Chapter three in this thesis also presents an optimisation approach using both the genetic algorithm and genetic programming to select the optimal parameters of a Hadoop cluster to enhance the performance of Hadoop jobs. The presented work uses both algorithms to obtain the optimal settings of the Hadoop parameters in conventional network and then, apply same parameters for the Hadoop cluster with different size in SDN environment to explore the impacts of the optimised parameters on the networking side of the cluster that can affect the performance of the execution time of Hadoop jobs. The complicated correlation between Hadoop parameters are taken into account through the optimisation process. Furthermore, some other ways have been also implemented on the resource provisioning to improve the performance of a Hadoop cluster [28] [29] [30]. The work in [31] optimised the performance of the Hadoop cluster by improving

the execution time of Hadoop MapReduce jobs. However, this model only supports specific type of jobs that based on the query and analysis for the short jobs. Piranha is another system that proposed in [32] to improve the performance of short jobs in Hadoop. The proposed work improved the response time of short queries by reducing the execution time of the running jobs. Similarly, SHadoop [33] is also proposed to optimise the performance of short jobs by improving their own execution mechanism. The setup and clean up tasks of Hadoop MapReduce jobs have been optimised in this work to reduce the cost time during the initialisation and termination phases. In the proposed work, instant messaging communication mechanism is also introduced to speed up the performance of task execution and scheduling.

2.5 Commercial Hadoop Distributions

The demand for big data technologies is increased due to the increasing number of connected devices that generate huge amount of data every day. Apache Hadoop became the core technology of big data applications because of its scalability and cost effectivity. It can be downloaded and installed for free from http://hadoop.apache.org .Since the open source of Apache Hadoop has some limitations, a number of commercial distributions vendors, such as Cloudera, Hortonworks, Amazon elastic MapReduce, MapR and IBM open platform are implemented to tackle them by providing additional features and functionalities for Hadoop platform users. Different functionalities and characteristics, such as reliability, completeness and support have been added to enable users to perform different tasks and applications using a Hadoop cluster [34]. In the following section, we provide an introduction of Apache Hadoop ecosystem.

2.5.1 Apache Hadoop Ecosystem

Hadoop is an open source framework that provides storage, processing and analysis for huge datasets. However, it is not a comprehensive system that can perform various applications of massive datasets. It is an efficient platform to process and analyse unstructured datasets stored in HDFS only. The storage and processing of other types of

datasets, such as semi structured and structured data is infeasible. Hence, Hadoop ecosystem has been developed to support Hadoop framework to perform many big data applications. The ecosystem of Hadoop contains several components as shown in Figure 2.5. A discussion for a list of Hadoop ecosystem components for Cloudera open source distribution including Apache Hadoop (CDH) is given below.

Figure 2. 5: Hadoop ecosystem in Cloudera distribution

Hue: hue is an open source web interface licensed under Apache 2.0 written in python and used by Apache Hadoop to support and manage its ecosystem. It can be installed on any PC as a web interface platform for big data analysis using Apache Hadoop. Hue supports many applications that provide web-based access to CDH components. Hue applications are managed by Hue server which interfaces and communicates with CDH components and other servers. Hue server acts as an intermediate container of web applications between CDH that are installed in PC and the web browser [35]. It is employed as editor for some applications of Hadoop ecosystem in CDH such as Hive, MapReduce, Impala, Pig, Spark and many more. It also can be used as a dashboard to visualise and explore data easily by Apache Solr and without any need for SQL programming. It automatically schedules the workflows and jobs as well as performs some operations, such as stopping or pausing jobs, job resubmission and job progress

monitoring. Furthermore, we present the browsers of Hue that is used to search and accomplish some operations on jobs and data in a cloud or a cluster network [36].

1- It browses yarn and Oozie jobs.

2- It browses tables such as (Hive, Impala, HBase).

3- Browsing for HDFS S3 and ADLS.

4- Browsing solr indexes.

5- The browsing of Sqoop and sentry policies.

Hive: hive is an open source software project written in java licensed under apache 2.0. It is built on top of Hadoop as a data warehousing for querying, summarising and analysing vast amounts of datasets that are stored in HDFS. It was initially developed by Facebook and it is also developed and used by different organisations like Netflix and the Financial Industry Regulatory Authority (FINRA) [37]. Hive uses query language called HiveQL, which is similar to SQL to enable SQL-like queries and convert them into MapReduce jobs to be executed in a Hadoop environment [38] . It consists of several parts that enable some operations on data in Hadoop ecosystem. A brief introduction about the main parts of Hive is given below.

1- Metastore: it stores the metadata for each table in Hive and support the driver to monitor the progress of different datasets that stored and distributed over different nodes in the cluster.

2- Driver: it controls and manages the lifecycle and progress of HiveQL statement execution.

3- Compiler: it performs the compilation process of HiveQL to convert the query from an abstract syntax tree (AST) to a directed acyclic graph (DAG).

4. Optimiser: it is responsible to conduct the transformation process on the execution plan to achieve high performance.

5- Executor: it executes and schedules the tasks through the interaction with the job tracker in the master node of a Hadoop cluster.

6- Hive server: Hive or thrift server provides a thrift user interface for external users to communicate with Hive through DBC/ODBC protocols.

Pig: pig is a high level language tool that is designed to query, analyse and process massive amount of datasets in parallel and distributed environment using a scripting language called pig Latin, which is similar to SQL in terms of data grouping, ordering and filtering. It uses the data flow process to represent and manipulate the huge datasets in a Hadoop cluster [39]. It is first developed by yahoo for analysing and processing large datasets by executing MapReduce jobs internally across several machines in the Hadoop cluster and store the output files in HDFS. It is now being used and developed by Apache software project to support many applications of huge datasets in the Hadoop environment. Pig provides an easy programming environment for programmers to execute MapReduce jobs using pig Latin. It also provides optimisations to support the automatic execution of tasks and provide high extensible to create special function for special purpose processing. However, it is only designed to support the batch processing mode of large datasets.

Oozie: Apache Oozie is an open source reliable and scalable tool, which built in java and designed to control and manages the workflow scheduling of MapReduce and pig jobs in distributed environments like Hadoop. It can process multiple jobs by combining them in a sequential order and run them concurrently using workflow definitions, which are written in hPDL (XML process definition language) [40] [41]. The workflow system is a collection of control flow and actions like MapReduce and big jobs that arranged in Directed Acyclic Graph (DAGS) that specify a sequence of actions to be executed. The control flow in Oozie performs some workflow operation, such as start, end, node failure and control the execution path of workflow.

HBase: HBase is an opens source scalable distributed database that written in java and developed by Apache software foundation to be run on top of HDFS. It is widely used by various enterprises, such as Facebook, Yahoo, Adobe, Netflix, Airbnb, Spotify and many more. It is developed based on the idea of Google Big table [42]. It was designed

to provide random and rapid real time access for Hadoop applications to read or write large data in HDFS with high throughput and low latency. HBase can store huge amount of structured data that includes billion of column and row pages in tables across different cluster nodes. It leverages the scalability and fault tolerant features to execute MapReduce jobs on these tables to extract useful information [43] [44]. Unlike relational database management system (RDBMS), HBase is not designed to support SQL queries and transactional applications. However, it is linearly scalable and able to store and distribute large number of tables through several cluster computing nodes unlike RDBSM, which is not scalable system and designed to store small number of tables. The manipulation process of large tables is managed by several components of HBase. For instance, the region server in HBase handles all tables that divided into several regions and processes read/write request for each region. It also configures and specifies the threshold size of each region. The region server can be run on each data node in the Hadoop cluster. The load balancing of all regions is managed by the HBase Master Server that assigns them to the region servers. The HBase master is also responsible to monitor and maintain the status of the Hadoop cluster. HBase can be scaled to be run on several servers; however, these servers are only managed by ZooKeeper that provides reliable distributed synchronisation and centralised management. A discussion about ZooKeeper is given below.

ZooKeeper: ZooKeeper is open source software project that written in java and licensed under Apache 2.0. It was originally developed by yahoo and now used by Apache software foundation in the Hadoop ecosystem to provide centralised and distributed configuration services for distributed applications. Many enterprises such as Yahoo, Reddit, Rackspace and eBay use ZooKeeper because of its reliability, scalability, high availability and simplicity. It can provide different services including configuration information management, distributed synchronisation, coordination services and naming services [43] [45] [46] .

Flume: Flume is a reliable, resilient and distributed service that built in java and licensed under Apache 2.0. It is also scalable and fault tolerant tool, which designed to collect and aggregate large streaming datasets from various sources such as log and

event files and move them into HDFS at higher speed [47]. Flume can efficiently import large event data that generated by various social websites like Facebook and twitter into HDFS or HBase. The data generated from Facebook and twitter or any other various sources is collected by several flume agents, which are number of JVM daemon process. Additional agents are also applied as a data collector, which is responsible to collect the data from the flume agents and move it into centralised storage like HDFS or HBase. Each flume agent mainly consists of three interactive components which play a significant role to collect, import and ingest the streaming event data into the Hadoop system. The first component of a flume agent is the sink, which receives the event data produced by the generators and transfers them to a channel which is next component of the flume agent. The channel is used as buffer storage of the event data received from the sink. Different channels can be used such as JDBC, file system and memory channel. The event data is consumed from the channels by a sink which transfers the consumed data to the following destination [48] [49].

Sqoop: Apache Sqoop is an open source tool built in java and designed to transfer massive datasets between Apache Hadoop and relational database systems. The name of Sqoop is derived from SQL and Hadoop. The structured datasets that stored in relational databases, such as MySQL, Postgres, Teradata and oracle are moved to Apache Hadoop to be processed and stored into HDFS and HBase using Sqoop command-line interface. Sqoop can be employed to perform some operations like ETL processing and data extraction from Hadoop. It can increase the system utilisation and improve the performance of parallel data transfer. It also improves the efficiency of data processing and analysis in Hadoop cluster by combining both structured and unstructured data [50] [51].

2.5.2 Hadoop 2

With the wide development of big data applications, Apache Hadoop launched another version, which is the Hadoop 2 to support different applications that cannot be achieved using Hadoop 1. Since Hadoop 1, supports only the batch processing of big data and lacks the support of non-MapReduce tools, Hadoop 2 was developed to support non-

batch processing and different applications of big data such as Machine learning applications in a distributed processing environment. It also provides the support of other distributed tools, such as spark and HBase to be implemented in Hadoop MapReduce environment. Hadoop 2 provides (YARN), which is the resource manager that manages the cluster resources, while data processing is performed using different processing tools. On the other hand, in Hadoop 1, the processing and the management of the cluster resources are both performed by the MR itself. One of the significant characteristics of the Hadoop system is the scalability. Despite both Hadoop 1 and Hadoop 2 are scalable systems; however, the limited number of computing nodes in cluster network of Hadoop 1 is the key limitation to process and analyse very large datasets in the cluster network. Therefore, developers have designed Hadoop 2 to support large number of machines to be run per cluster. In Hadoop 2, the number of cluster machines can be scaled up to 10000 nodes per cluster while in Hadoop 1; the cluster network can run up to 4000 nodes [52] [53]. Hadoop 1 has several data nodes and one Name Node that manages the namespaces of the entire cluster network. Hadoop 2 provides HDFS federation to address the drawbacks of previous HDFS architecture that supports only single Name Node to manage the entire cluster. Multiple Name Nodes can be used with HDFS federation to improve the scalability of the name services and support the management of the whole cluster namespace [54]. It provides a remarkable feature for HDFS architecture by configuring two name nodes in an active/passive mode instead of one name node to avoid the Single-Point-of-Failure (SPOF) and provides high availability (HA) of HDFS. This feature has been added to overcome the limitation in Hadoop 1 where a failure happens in the single name node, operator intervention is required to overcome such issue. On the other hand, standby name node is added in Hadoop 2 to automatically overcome any crashing hardware or software in the active name node.

2.5.3 Hadoop 3

Hadoop 3 has included some important improvements over the previous editions of Hadoop 1 and Hadoop 2 [55]. The minimum support of Java has been increased from Java 7 to Java 8 in Hadoop 3. The fault tolerant feature and the storage scheme in the Hadoop 3 has been improved by enabling erasure coding in HDFS to reduce the storage

overhead instead of using replication method in both Hadoop 1 and Hadoop 2. Hadoop 3 has improved the scalability and availability of the Hadoop system by adding multiple standby name nodes. The timeline service of Hadoop 2 has been updated to be more scalable and reliable in Hadoop 3. It can improve the performance of machine learning and data science applications by enabling GPUs support that play a significant role to enhance the computations needed for these applications. It performs better in data balancing than Hadoop 1 and Hadoop 2, because it supports intra-data node disk balancing. The HDFS balancing scheme used in Hadoop 1 and Hadoop 2 is not effective balancer, because it leads to unequally disk space in each server in case of adding new storage to an existing server with old drives. Therefore, applying intra-node disk balancer can overcome this issue and make the disk space in each sever is distributed equally [56] [57].

2.6 Machine Learning Applications in Hadoop Environment

Nowadays, massive datasets are generated from different sources and applications. Extracting and predicting useful information based on input data and without any explicit programming has become possible with machine learning algorithms. However, the prediction and extraction of such information from huge datasets is a challenging task and time consuming, because it requires high computation resources, which are not provided by single machines. As a result, some software and tools have been designed to support the integration between machine learning algorithms and a Hadoop framework that supports the implementation of several machine learning applications and techniques using multiple cluster machines. An introduction of some machine learning tools in Hadoop environment is given below.

Apache Mahout: Mahout is an open source distributed framework that was developed to support machine learning applications running on top of a Hadoop cluster. It can run scalable machine learning algorithms based on Hadoop platform using MapReduce paradigm to implement different machine learning techniques such as clustering, filtering, classification and recommendation [58]. It is also used to perform different machine learning applications like forecasting, data mining, pattern recognition and

many more. It provides some math and statistic operations by applying java libraries. Furthermore, Mahout supports applications that require evolutionary programming by providing distributed fitness function capabilities. It uses the library of Apache Hadoop to scale the machine learning algorithms to be run in the cloud computing environment. Moreover, it can speed up the execution time of analysing large datasets for different machine learning applications. Some clustering algorithms, such as k-means, fuzzy k-means and mean-shift can be implemented using Apache mahout based on the MapReduce programming. Many companies and organisation such as Facebook, Twitter, LinkedIn, Adobe and yahoo use Mahout for different needs.

R language: R is a programming language and free software tool that is written in C, Fortran and R. It was first developed by Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand and it is currently developed by the R development core team. R is mainly designed to perform calculations and statistical computing on data. It is also widely used for time series analysis, data mining, machine learning applications (linear and nonlinear modelling, classification and clustering) [59]. R Language provides reliable and flexible services for data manipulation, calculation and analysis such as data cleaning, data extraction and predictive modelling. It can effectively store data as it can connect with different data stores such MySQL, MongoDB and HDFS in Hadoop cluster. It also can handle data and provides effective support for relational database systems. Efficient tools for data analysis and visualisation are also provided by R language. It is not a statistical tool but provides the efficient environment to implement different statistical techniques to extract meaningful information from datasets. However, it is not scalable and distributed system that can handle large scale datasets, it is designed to handle and process limited amount of dataset. As a consequence, it became imperative step to integrate R language with Apache Hadoop, which is the main platform for big data processing and analysis. The operations of data analytics with R language can be scaled up using several computing machines running on a Hadoop cluster. The following section explains the integration between R and Hadoop.

RHadoop: RHadoop is an open source framework developed by revolution analytics to support big data analytics with R based on a Hadoop cluster [60]. It has three different collections of R packages including rhdfs, rhbase and rmr. All three packages have been implemented and tested on recent versions of Cloudera and Hortonworks commercial Hadoop distribution as well as the recent R version [61]. The connection between HDFS and R is provided by rhdfs, which allows R users to read, write and amend any files that managed by R functions and stored in the HDFS. On the other hand, the files stored in HBase distributed system can be read, write and modified by R programmers using rhbase. Map and Reduce functions can be applied on different statistical analysis in R using rmr software package.

RHIPE: RHIPE is an open source software package that used for big data analytics using R expressions and MapReduce programming in a Hadoop environment [62]. It was developed by Mozilla's Suptarshi Guha at Purdue University in 2012 to enable R programmers to run Map and reduce jobs in R environment based on Hadoop and HDFS. It uses D&R mechanism (divide and recombine) to perform the analysis on large datasets in a distributed environment. It takes the large input datasets and divides them into several subsets and then applies R operations within the map phase to process them in parallel to obtain intermediate results. All the intermediate results that generated from the map phase are recombined during the reduce phase into a set to obtain the final output. Since RHIPE is an integrated system of R and Hadoop, several software packages and libraries need to be installed on several machines in the cluster network. The installation process of RHIPE is explained in chapter 4.

2.7 Data Centre Network Architecture

Data centre network (DCN) is a collection of large number of interconnected devices that provide computational, storage and network services for different users and enterprises. It includes a large number of servers and switches, which are connected through communication links. It has been widely deployed for a wide variety of internet and cloud-based applications like Google, Yahoo, YouTube and many social networking including Facebook and twitter. It is also used for data processing and

management by various projects, such as Google file system (GFS), Big Table, Dryad, Hadoop distributed file system (HDFS) and MapReduce programming model. One of the largest data centres in the world is Chicago-based data centre that provides different Microsoft services. With the increasing demand of data centre network in different cloud-based services, large number of hosted servers has been increased to provide efficient computing and storage services. However, high traffic is generated due to the information exchanged between the large numbers of servers that stresses the bandwidth in traditional data centre network. As a result, some architectures have been proposed to tackle the traffic issues and improve scalability, resiliency, flexibility, fault tolerance, cost effectiveness and energy efficiency [63] [64]. One of the proposed network topologies of a data centre network is fat tree topology that was proposed by Al-Fares [65]. A description of fat tree topology is given below. Furthermore, leaf-spine topology is another architecture, which is explained in section 2.7.2.

2.7.1 Fat Tree Topology

The traditional design of a data centre network involves three different layers of interconnected switches or routers. Fat tree topology consists of three layers of switches. The first one is the core layer that provides packet forwarding services for all flows between the lower layers within the data centre network as well as flows going outside the data centre. This layer provides resilient route to avoid single point of failure and provides connections between different aggregation layer switches. The second one is the aggregation layer, which is located in the middle between the core and edge layer. It has some integrated modules that provide various services, such as firewall, network analysis, intrusion detection, SSL offload and server load balancing. The edge layer, which is located on the top of rack, is the third layer that provides the connection between different hosted servers within the same rack. The hosted servers in the data centre network can directly connect to the edge layer switches. The switches in the core layer provide the connection between the data centre and the internet [66] [67]. Fat tree topology has some salient features that can be invested to obtain efficient performance for big data applications like MapReduce in the data centre network. Fat tree topology has a number of pods depending on the topology size of the data centre network. Each pod contains a number of edge and aggregation switches that connect to

each other without the need to pass through the switches in the core layer. However, if the switches in the aggregation and edge layers are located in different pods, the traffic between them should pass through the switches of core layer. According to [68] the relationships between the switches in the same pod or in different pods should be exploited to achieve effective routing and better path allocations for the flows in the data centre network. Moreover, another significant feature, which is the connection between the hosted servers, can be also employed to obtain an efficient path allocation. Three different connections are exist among the servers in the fat tree topology, the connection between any two servers in the same rack called rack-local connection, because the traffic between them remains inside the rack. On the other side, the connection between two servers, which are located within the same pod called pod-local connection as the traffic between the two connected servers should traverse at least one switch in the aggregation layer. The last type of connection between two servers is the remote connection where both two servers are located in different pods. In this connection, the traffic between the two servers should traverse the switches in the core layer. The architecture of fat tree topology is shown in Figure 2.6.

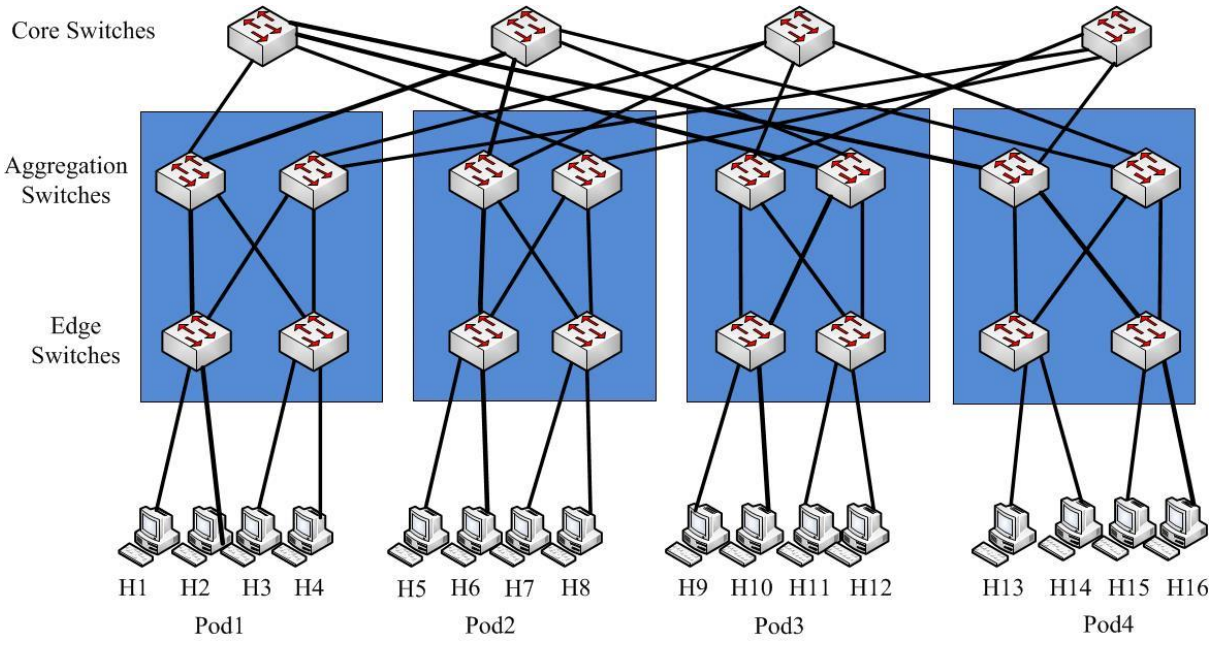

Figure 2. 6: fat tree topology architecture

2.7.2 Leaf-Spine Topology

Due to the wide use of cloud networks and data centre network, efficient and resilient design is highly required to improve the performance of east-west traffic. As a consequence, a flat design like leaf-spine is proposed to meet the requirements of current network applications, such as cloud applications and big data applications. Leaf-spine topology is a list of leaf switches at the access layer, which are connected to a list of spine switches at the top layer in a mesh full topology. Each leaf switch is connected to spine switch; however, the switches in the same layer are not interconnected to each other as shown in Figure 2.7. One of the main advantages of leaf-spine topology is the minimization of latency and some other bottlenecks between the leaf switches by making only one hop away from each other. Both layer 2 and layer 3 can be used to switch and route the links between the leaf and spine switches. In the layer 3, each link between the spine and leaf layers is routed using open shortest path (OSPF) or border gateway protocol (BGP) based on equal cost multiple path (ECMP). On the other hand, Transparent Interconnection of Lots of Links (TRILL) or Shortest Path Bridging (SPB) can be applied by layer 2 for switching the links in the leaf-spine topology [69].

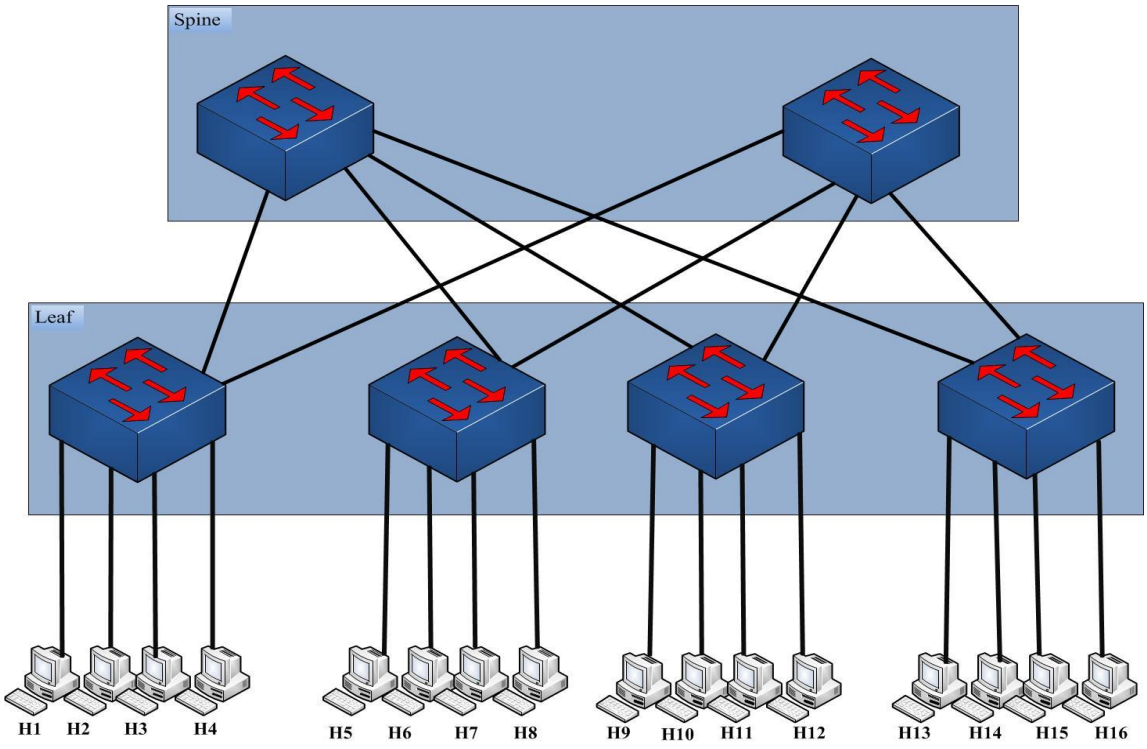

Figure 2. 7: leaf-spine topology architecture

2.7.3 Routing Algorithms in Data Centre Network

Several algorithms are exist for general routing in computer and communication networks. These algorithms can also be applied to rout the traffic of shuffling flows between different hosts in a data centre network based on fat tree topology to improve the performance of big data applications. For instance, the traffic of shuffling flow between any two hosts can be routed by selecting the shortest path between them based on the shortest path algorithm. However, the shortest path might trigger congested links in the fat tree topology. Single best path for the traffic can be selected using Open Shortest Path First algorithm (OSPF) [70]. However, this algorithm lacks the efficient scalability. Since the fat tree topology provides redundant links, Equal Cost Multipath Algorithm (ECMP) [71] can now be used to route the traffic using these links to utilise multiple paths between any two hosts to route the traffic of shuffling flows. However, use multiple paths will result in a low performance especially for large data centres, because the amount of entries in the routing table increases exponentially and consequently increase the latency of the routing algorithm. In addition, the nature of conventional data centre network can also affect the performance of routing algorithms for flow scheduling due to its static network configurations. The flow scheduling can only be implemented by the switches in the aggregation and core layers because of the hardware constraints. As a result, a central controller is required to achieve better flow scheduling. The solution is provided by software defined network that can provide a dynamic network configuration to adopt different requirements for different applications.

2.8 Cloud Computing Technology

Cloud computing is a term that refers to use wide variety of computing services, storage and other IT resources over the internet. In recent years, cloud computing has become very common technology that used by many companies and organisations. Many companies are choosing to depend on the cloud computing technology for crucial business duties. Some companies, such as Google and IBM use cloud computing to store their information and manage their resources. Furthermore, cloud computing is used in different fields including marketing, data centres and libraries [72]. It is also

scalable and reliable technique for distributed processing and massive data sets storage. Amazon uses cloud computing in the form of virtual servers to provide elastic computing services and storage management for big data applications, such as MapReduce and other distributed tools that run on top of apache Hadoop. Due to many facilities offered by cloud computing, a large number of companies have depended on it and many of them have invested more money on it recently. According to Gartner research, the expectancy of investment in cloud computing will reach $150 billion. A survey has been conducted on six data centres showed that between 10% and 30% of computing power was utilised by servers, while less than 5% of the computing power was employed by Desktop PCs. About 66% of the information technology budget is spent by companies and organizations on support and conservation activities, which are not needful in the era of globalisations [73]. A phone survey of 54 organizations was conducted in the summer of 2012, ranging in size between twelve and hundred thousand of workers. According to this survey, some form of cloud service was being used by 22% of users. The average size of cloud services users was 1,378 workers, in comparison with the average size of 322 workers who were not utilising the cloud services. Today, the larger companies and enterprises are exploring methods to include expenses and manage the enormous increasing of data sets [74].

Cloud computing has salient characteristics that attract many enterprises and organisations, such as scalability, resiliency, on-demand utilisation, ubiquitous access and cost effectivity. It provides many benefits for many companies and users. It has become easily for users to use and access servers, storage and databases over the internet using cloud web services platform. For instance, as we mentioned before amazon can offer reliable computing resources for users and organisations to process, store and analyse large scale data sets through virtual computing and distributed environment called instances. The companies and users can choose their own instances, according to their processing and storage requirements. There are several types of cloud services are provided like public cloud, where the services of this cloud are open to the public and can be accessed by users through the internet; such services are provided by Google and Microsoft. Some companies prefer to choose their cloud services to be private where they can store and manage their own data that cannot be accessed or shared with other companies or organisations. This type of cloud services can be managed by the company itself or third party. Some other companies use the hybrid

type of cloud services where both public and private cloud services are combined together. These companies choose to store their own critical information on the private cloud, whilst the non-critical information is stored on the public cloud [75]. Cloud computing has three main models [76] [77] ,which are explained below.

Infrastructure as a service (IaaS): IaaS provides the hardware infrastructures of cloud computing including virtual serves, computing facilities, storage system and database management, network devices and data centre place. Users can configure and install any software and write some programs over the services provided by Iaas. Some companies that provide such services are Amazon web services, GoGrid and 3 Tera.

Platform as a service (PaaS): PaaS is a software layer that runs on infrastructures provided by Iaas. It allows customers to manage and develop their own applications without the need to build and maintain the infrastructures by providing operating system and application servers through platform providers like Google App engine, force.com, Azure service platform and LAMP platform (Linux, Apache, MySql and PHP).

Software as a service (SaaS): SaaS is software provided to the users over the internet as a service on demand. It can be accessed by the users through a web browser. Using SaaS, the users are not required to maintain storage space, handling data loss and manage installations for software as all of these services are provides by SaaS. It is provided by many companies like Google Docs, Salesforce, Microsoft and many more. Figure 2.8 shows the three models of cloud computing.

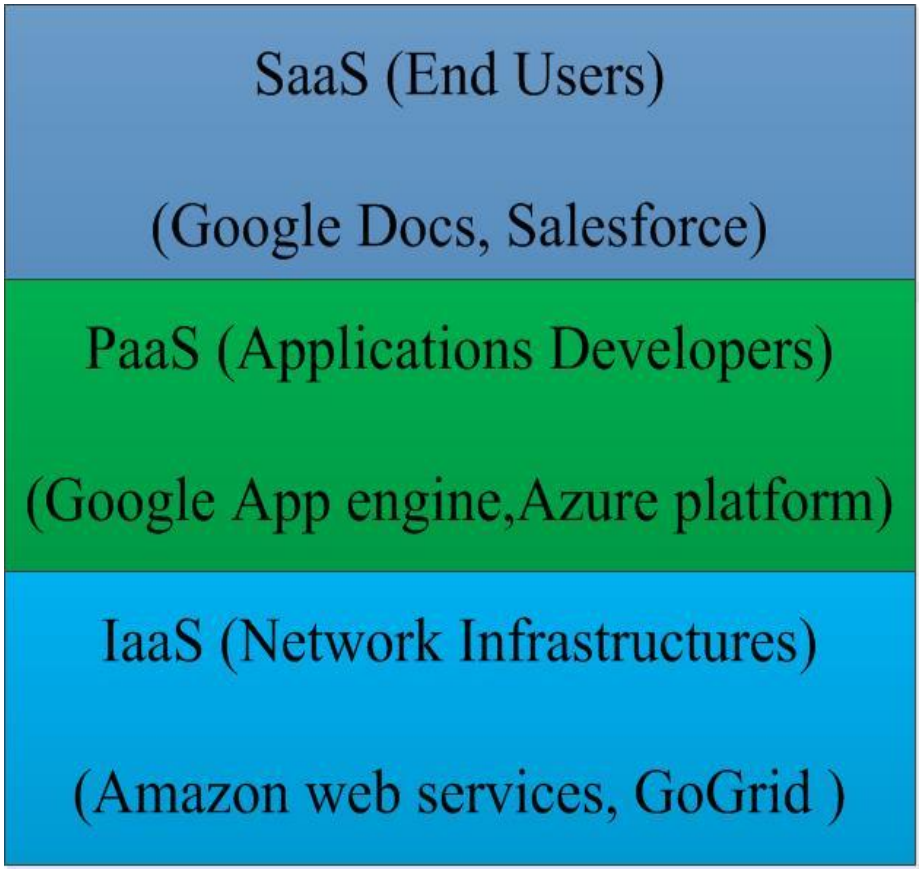

Figure 2. 8: Cloud Computing Model

2.9 Software Defined Networking

With the advent of internet of things (IoT) and virtualisation technology of IT and networking resources as well as the increasing demand of cloud computing services, it has become very crucial for networking industry to explore efficient network architectures that can cope up with the increasing size of networks due to the large number of connected devices. Software defined networking (SDN) is a novel technology that can provide the solution for many network applications that cannot be managed by the conventional IP networks. It is the technology that separates the control plane from the data plane to tackle the limitations of current network infrastructures by providing agile centralised management and intelligent programming for network configurations. In conventional networks, both the forwarding and routing process are performed by network device itself. However, in SDN environment, the control plane is separated from the data plane to be implemented as a software layer that runs on commodity server called the SDN controller. The control plane is responsible to

manage and control the routing process of packets in the SDN network. It acts as the brain of the SDN network that provides an intelligent and centralised management. On the other hand, the forwarding process of packets in the SDN network is performed by the data plane (network device). Software defined network brings many benefits that can have a significant impact on many network applications. It provides the support for the accurate and automatic configuration management of the entire network. It also supports automated load balancing and the scalability of network resources by providing data flow optimisation technique that enables multiple paths for each flow from the source to the destination. This technique, which is provided by the SDN controller, can split the traffic of each flow across multiple nodes in the network. SDN can also optimise the networking and computing resources, in addition to the storage resources in the network. It is able to provide efficient network administrations and improves the performance of server virtualisations that leads to reduce the operation cost [78] [79]. The architecture of SDN consists of three layers [80] [81], which are explained below.

1- Application layer: Application layer is the layer of SDN applications, which provides programming interfaces where programs directly communicate and exchange information with the SDN controller via northbound APIs. This layer is responsible to provide abstract network view of network applications by collecting status information like network devices and links from the SDN controller. Information collected can help the control layer to obtain efficient guidance for the network configurations.

2- Control layer (SDN): Control layer represents the core of the SDN network that includes the control plane, which is responsible to manage and program the data plane to provide the routing operations of the network. The flow control between the software controllers in this layer can communicate with the forwarding (data plane) via southbound APIs and with the SDN applications of application layer via northbound APIs to provide intelligent networking operations.

3- Infrastructure layer: Network infrastructure layer is also called the data plane layer, because it manages and controls the forwarding process of data path. It includes the SDN forwarding devices that perform the forwarding and routing decisions based on the flow tables provided by the SDN controller in the control layer via the southbound APIs. The layers of the SDN network are shown in Figure 2.9.

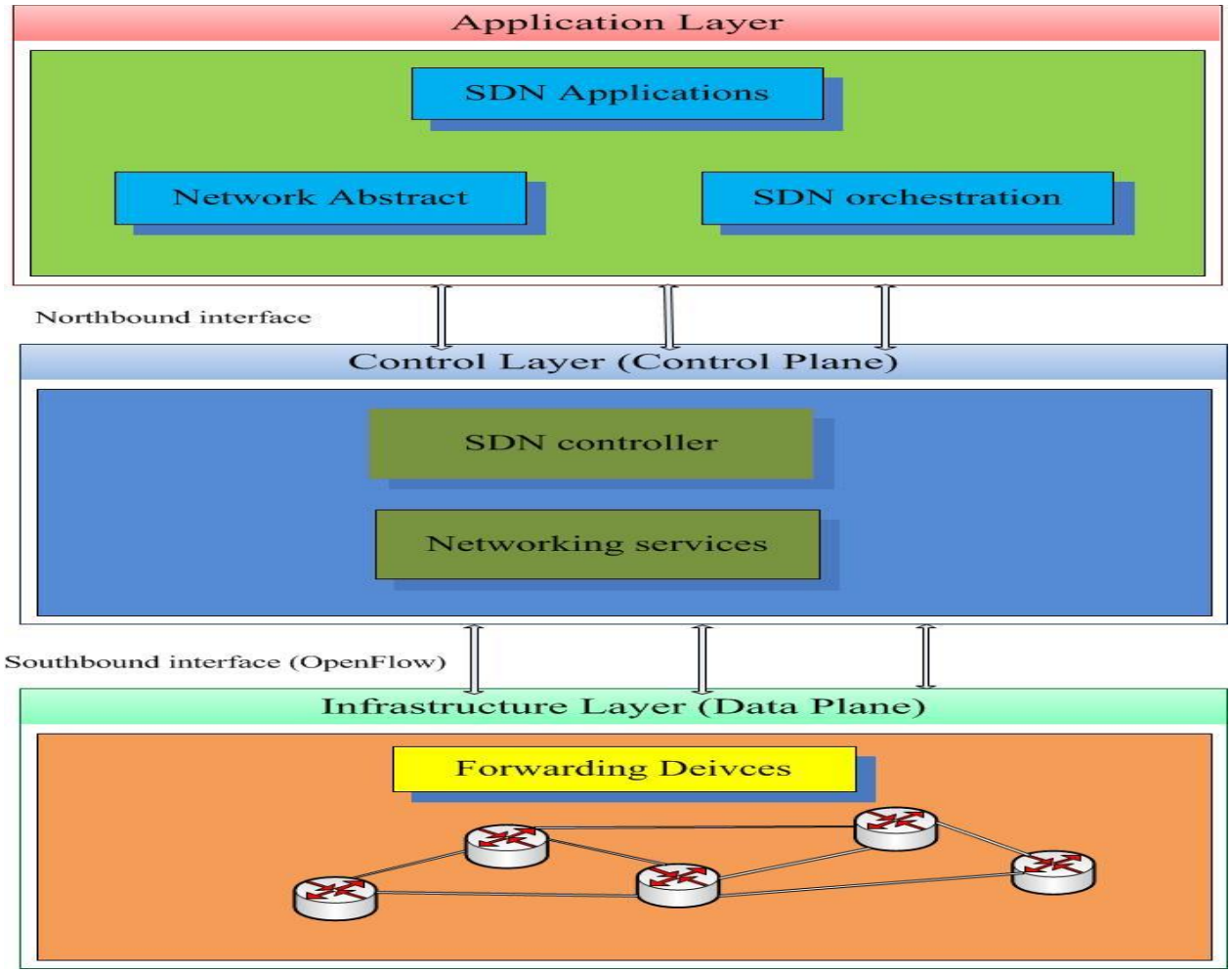

Figure 2. 9: SDN layers architecture

2.9.1 OpenFlow

OpenFlow is a standard network protocol provided by the Open Networking Foundation (ONF) for software defined networking. It acts as an interface between the control layer (control plane) and the infrastructure layer (data plane). OpenFlow enables the interaction between the SDN controller in the control plane and the forwarding devices (OpenFlow switches) in the data plane. All switches in the SDN network are OpenFlow switches that forward traffics according to the flow table entries installed by the SDN controller. OpenFlow supports both physical and virtual switches that work in hypervisor mode. The traffic in the SDN network is identified using flows, which are sequence of packets transferred from the source to the destination that have the same

value in particular fields over the OpenFlow switches. Each packet is matched based on the match rules specified in the flow table that provided by the SDN controller [82].

2.9.2 Floodlight Controller

Floodlight controller is a SDN controller that licenced under Apache and written in java. It is provided by big switch networks to support OpenFlow switches in software defined network. The OpenFlow switches, which are connected to the floodlight controller, are managed by the floodlight module. The floodlight controller can be easily used and configured. It can also develop applications that are written in java by providing REST APIs. It supports combination of the SDN network and the conventional network [83] . It is widely used by researchers, because it is open source and can support both physical and virtual switches. Since it supports the virtual switches, researchers can test and develop their own models according to their needs in virtual environments. Floodlight controller provides CPU efficiency and supports multiple threads due to its java nature. It also provides efficient memory management and can be run on several platforms [84].

2.9.3 Hadoop Networking Performance Improvements Techniques

With the emergence of big data concept, Hadoop framework has become one of the main platforms for many big data applications. However, some issues regarding the networking aspect of a Hadoop cluster have been identified. MapReduce programming techniques are used to analyse and process massive data sets across several computing machines in the Hadoop cluster. Since the MapReduce consists of two phases, which are the Map and Reduce phase. The transfer process of the intermediate data that is generated from the map phase to the reduce phase goes through the shuffling phase. High traffic is generated during this phase to transfer the intermediate data of the map phase to the reducers that communicate with other nodes in the cluster to collect the intermediate results. Moreover, another traffic is also generated during the reduce output phase when the final outcomes of the MapReduce job are written to the HDFS with three replicas depends on the replication factor that configured in the Hadoop

cluster. The situation becomes more sophisticated in large scale clusters, especially in the data centre network where large numbers of switch devices are included. Consequently, several research works have been proposed to discuss and tackle such issues to improve the performance of Hadoop jobs by reducing their execution time. Most of the proposed systems used Software defined networking to enhance the networking side of a Hadoop cluster with different topologies. Some works like [85] proposed bandwidth-aware scheduling for tasks allocation on local data nodes. In the work, heuristic bandwidth-aware task scheduler has been proposed based on SDN to assign tasks in a global view and improve data locality. Similar work have been also implemented such as BAlance-Reduce(BAR) to improve data locality in an optimised global way [86]. Some other works have focused on the network-awareness instead of bandwidth-awareness. Zhao and Medhi developed Application-Aware Network (AAN) platform based on SDN to improve the performance of a Hadoop MapReduce job without any modifications in the underlying design of Hadoop MapReduce framework [87]. The proposed system in [68] developed an application-aware routing algorithm based on SDN to speed up the execution time of the shuffling traffic for the MapReduce job in the data centre network. This thesis also propose routing algorithm based on SDN to improve the performance of a MapReduce job by reducing the execution time of the shuffling phase in the data centre network. The details of the proposed work are presented in Chapter 5.

2.10 Summary

In this chapter, the background of the Hadoop framework and MapReduce programming technique was presented. This chapter also presented the architectures of data centre network as well as the routing algorithms used to route the traffic inside it. SDN was also presented in details along with the open flow protocol and the floodlight controller. This chapter presented the cloud computing technology and the ecosystem of Hadoop framework. A number of optimisation techniques and methods used in optimising the computing aspect and networking of a Hadoop MapReduce job were also reviewed.

Chapter 3

Optimisation of MapReduce Configuration Parameter Settings Using Genetic Algorithms for Hadoop Cluster Based on Software Defined Networking

Nowadays, Hadoop framework with MapReduce programming technique has been widely used for big data analytics. Hadoop MapReduce is a complicated system that contains large number of complex parts and parameters. Hadoop MapReduce includes a large number of parameters which is set with default values. Some of these parameters can affect the overall performance of Hadoop MapReduce jobs. It can be improved by tuning theses parameters with the optimal settings. However, setting this number of parameters manually with the optimal settings faces some challenges from the aspect of time and accuracy.

3.1 Introduction

Big data is a term that refers to the large and complex data sets that cannot be processed, captured, stored and analysed using traditional tools [17]. These amounts of huge data are generated from different, sources such as social media, sensor devices, the Internet of things, mobile banking amongst many more origins. Furthermore, many governments and commercial organisations are producing large amounts of, data such as financial and banking statements, healthcare providers, high education systems, research centres, the manufacturing sector, insurance companies and the transportation sector. Regarding which, International Data Corporation (IDC) reported that 2,800 Exabyte of data in the world were stored in 2012 and this is expected to reach up to 40,000 Exabyte over the next ten years. For instance, Facebook processes around 500,000 GB every day. The vast amount of data includes both structured, such as relational databases as well as, semi structured and unstructured data, such as texts, videos, images, multimedia, and web pages. These types of huge data with various formats have led to the coining of the term big data [7].

However, these massive datasets are hard to be processed using traditional tools and current database systems. Hadoop MapReduce is a powerful computing technology tasked with supporting big data applications [18]. Hadoop is an open source framework that enables the implementation of the MapReduce algorithm for data processing purposes. It is scalable, fault-tolerant and able to process massive data sets in parallel. Moreover, large datasets can be distributed across several computing nodes of a Hadoop cluster to achieve better computation resources and power [88]. Hadoop has a complex structure that contains a number of parts that react with each other through several computing devices. Moreover, Hadoop it has more than 150 configuration parameters and recent studies have shown that tuning some of these can have a considerable effect on the performance of a Hadoop job [89] [13].

Because of the black box feature of the Hadoop framework, the tuning of parameters values manually is a challenging task as well as being time consuming. To tackle this issue, genetic algorithms for Hadoop have been developed to achieve optimum or near optimum performance of the Hadoop MapReduce parameter settings. However, there are some traffic issues for Hadoop jobs especially in the shuffling phase during the transfer of intermediate output data from the mappers to the reducers. As a consequence, SDN is proposed to alleviate these traffic issues in a Hadoop cluster. SDN was employed for a small Hadoop cluster using 14 virtual machines connected to one physical switch and two open virtual switches. SDN was also used to evaluate the performance of Hadoop jobs in a large scale cluster in a data centre network.

3.2 Related Work

Many ways have been proposed for the automatic tuning of Hadoop MapReduce parameter settings, one of which being PPABS [90] ((Profiling and Performance Analysis-Based Self-tuning). In this framework, the Hadoop MapReduce parameter settings are tuned automatically using an analyser that classifies MapReduce applications into equal classes by modifying k- means ++ clustering and a simulated annealing algorithm. Furthermore, recogniser is also used to classify unknown jobs into one of these equivalent classes. However, PPABS cannot tune parameters of an unknown job not included on these equivalent classes. Another approach, called Gunther, has been proposed for Hadoop configuration parameters optimisation using

genetic algorithm. However, all MapReduce jobs have to be executed physically to evaluate the objective functions of required parameters, because Gunther does not have an objective function for each of them. Moreover, the execution time for running MapReduce jobs for objective function evaluation is very long [91]. Panacea framework has been proposed to optimise Hadoop applications based on a combination of statistic and trace analysis using a compiler guided tool. It divides the search place into sub places and subsequently performs a search for best values within predetermined ranges[92]. A performance evaluation model of MapReduce is proposed in [93]. This framework correlates performance metrics from different layers in terms of hardware, software, and network. Industrial professionals proposed the Rule-Of-Thumb (ROT), which is merely a common practice for Hadoop parameter settings tuning[94] [95]. In [96] an online performance tuning system for MapReduce is proposed to monitor the execution of a Hadoop job and it tunes associated performance-tuning parameters based on collected statistics. [97] Optimises MapReduce parameters by proposing profile to collect profiles online during the execution of MapReduce jobs in the cluster. In [98] a self-tuning system for big data analytics, called starfish, is proposed to achieve the best configurations of a Hadoop framework so as to utilise cluster resources better in terms of CPU and memory.

3.3 Hadoop MapReduce Parameters Settings

Hadoop is a software platform written in java that enables distributed storage and processing of massive data sets using clusters of computer nodes. It provides large storage of any type of data (structured, semi structured and unstructured data) due to its scalability and fault tolerance. Furthermore, it is a complex system that contains a large number of components and parameters that interact witch each other. It has more than 150 tuneable parameters that play a vital role on the flexibility of Hadoop MapReduce jobs and some of them have remarkable influence on performance of Hadoop jobs if they are tuned with optimal values. In this work, we consider eight parameters that have a significant impact on the Hadoop jobs performance as shown in Table 3.1.

Table 3. 1: The main parameter settings of Hadoop framework

parameter	Default
MapReduce.task.io.sort.mb	100
MapReduce.task.io.sort.factor	10
Mapred.compress.map.output	false
MapReduce.job.reduces	1
Mapreduce.map.sort.spill.percent	0.80
MapReduce.tasktracker.map.tasks. maximum	2
MapReduce.tasktracker.reduce.tasks. maximum	2
Mapred.job.shuffle.input.buffer.percent	0.70

Below further description of the main parameter settings mentioned in the Table 3.1

1) MapReduce.task.io.sort.mb: During sorting files, amount of buffer memory is required for each merge stream. This amount is determined by this parameter and by default it is set to be 1MB for each merge stream and the total amount is 100 MB.

2) MapReduce.task.io.sort.factor: This parameter determines the required number of merged streams during sorting files process. The default value is set to be 10 as explained in Table 3.1.

3) Mapred.compress.map.output: The output results generated from mappers should be sent to the reducer through the shuffle phase. However, high traffic is generated during the shuffling process especially when the output data of mappers is large. Therefore, the results generated from mappers should be compressed to reduce the overhead in the network during the shuffling process and thus accelerate the hard disk IO.

4) MapReduce.job.reduces: a specific number of map tasks are required to perform the process of MapReduce job in Hadoop cluster. This number of map tasks is specified by this parameter. Default settings of this parameter are assigned to 1. Furthermore, this parameter has a significant effect on Hadoop job performance.

5) Mapreduce.map.sort.spill.percent: the default setting of this parameter is 0.80 which represents the threshold of in memory buffer used in map process. The data of in memory buffer is spilled to the hard disk once the in memory buffer reaches to 80%.

6) MapReduce.tasktracker.reduce.tasks.maximum : each MapReduce job has several Map and Reduce tasks running simultaneously on each data node in Hadoop cluster by task tracker. Reduce tasks number is determined by this parameter and its default setting is set to be 2. This parameter can have an important impact on the performance of Hadoop cluster when better utilising the cluster resources in terms of CPU and memory by tuning this parameter to the optimal value.

7) MapReduce.tasktracker.map.tasks.maximum: while number of reduce tasks is determined by parameter 6, this parameter defines number of map tasks running simultaneously on each data node. The default value of this parameter is 2. On the other hand, any change in the default settings of this parameter can have a positive impact on the total time of MapReduce job.

8) MapReduce.reduce.shuffle.input.buffer.percent: the output of mapper during the shuffling process requires a specific amount of memory from the maximum heap size for storage purposes. The percentage of this mount is determined by this parameter and its default value is set to be 0.70.

3.4 Evolving Hadoop MapReduce Parameters with Genetic Programming

Genetic programming (GP) [99] is a technique used to solve problems automatically with a set of genes and chromosomes. These are evolved using two essential genetic operations: crossover and mutation. In this work, GP is employed to create an objective function of the MapReduce parameters. The parameters of Hadoop MapReduce are represented as (*k1, k2, ……,kn*,) and here, eight parameters are tuned using a genetic algorithm (GA). An objective function should be built first using GP. Hence, a mathematical expression or function between these parameter settings needs to be determined. GP is used to evolve an expression between these parameters using arithmetic operations (*, +, ¬-, /). The fitness assigned to each parameter during the population process in GP should reflect how closely the output of the mathematical expression (function) for this parameter is to that for the original one. The objective function for GP is the execution time of running MapReduce jobs.

The tree in GP consists of two levels. The first one is called functions and used to hold operations. The second is called terminals or leaves and used to hold the input data. The arithmetic operations in GP are the functions, while the parameters (*k1,…,kn*) are the leaves of the tree. The mathematical expressions between the Hadoop MapReduce parameters are determined based on their data type. The mathematical expression should have same input data type and same number of input parameters. After its determination, the completion time of these functions needs to be calculated and compared with the real one. The best mathematical expression among the parameters (*k1,…,kn*) will be selected based on its approximated completion time, which should be very near to the real one. The tree in GP is used to hold both functions and terminals. As mentioned above, arithmetic operations (*, +, ¬-, /) are called functions and (*k1,…,kn*) are called leaves or terminals. Figure 3.1 shows an example of the representation of parameters using GP.

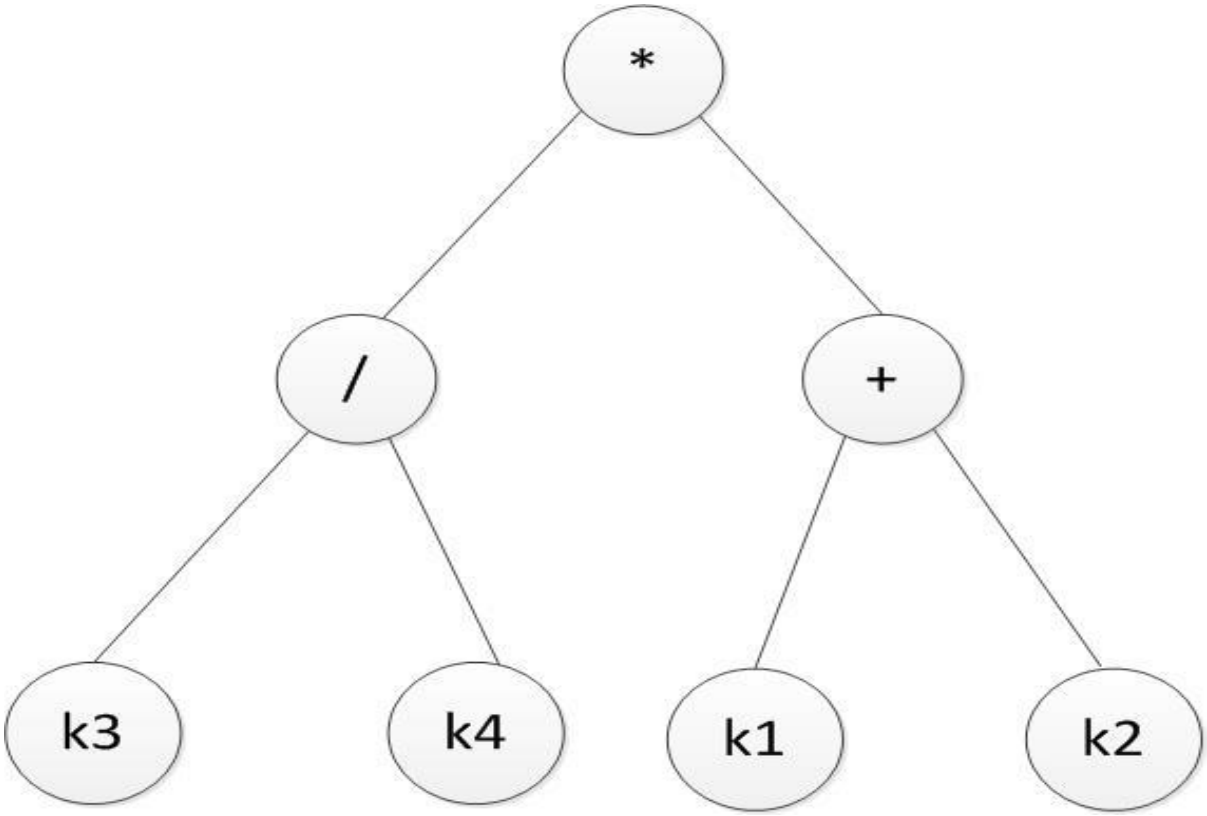

Figure 3. 1: an example of Genetic programming

The Figure shows that the function (*) has two input arguments, which are (+) and (/) and the function (+) also has two (*k1*, *k2*). The completion time of MapReduce job of Hadoop parameters can be represented as *f* (*k1*, *k2*,.., *kn*). The approximated completion time of Hadoop MapReduce job represents the evolved function that will be compared to the real completion time of Hadoop MapReduce that pertains to the target function. According to [99], the approximated completion time of Hadoop MapReduce (evolved function) should be very near to the real completion time of the job (target problem or function). Figure 3.2 shows the procedure of GP.

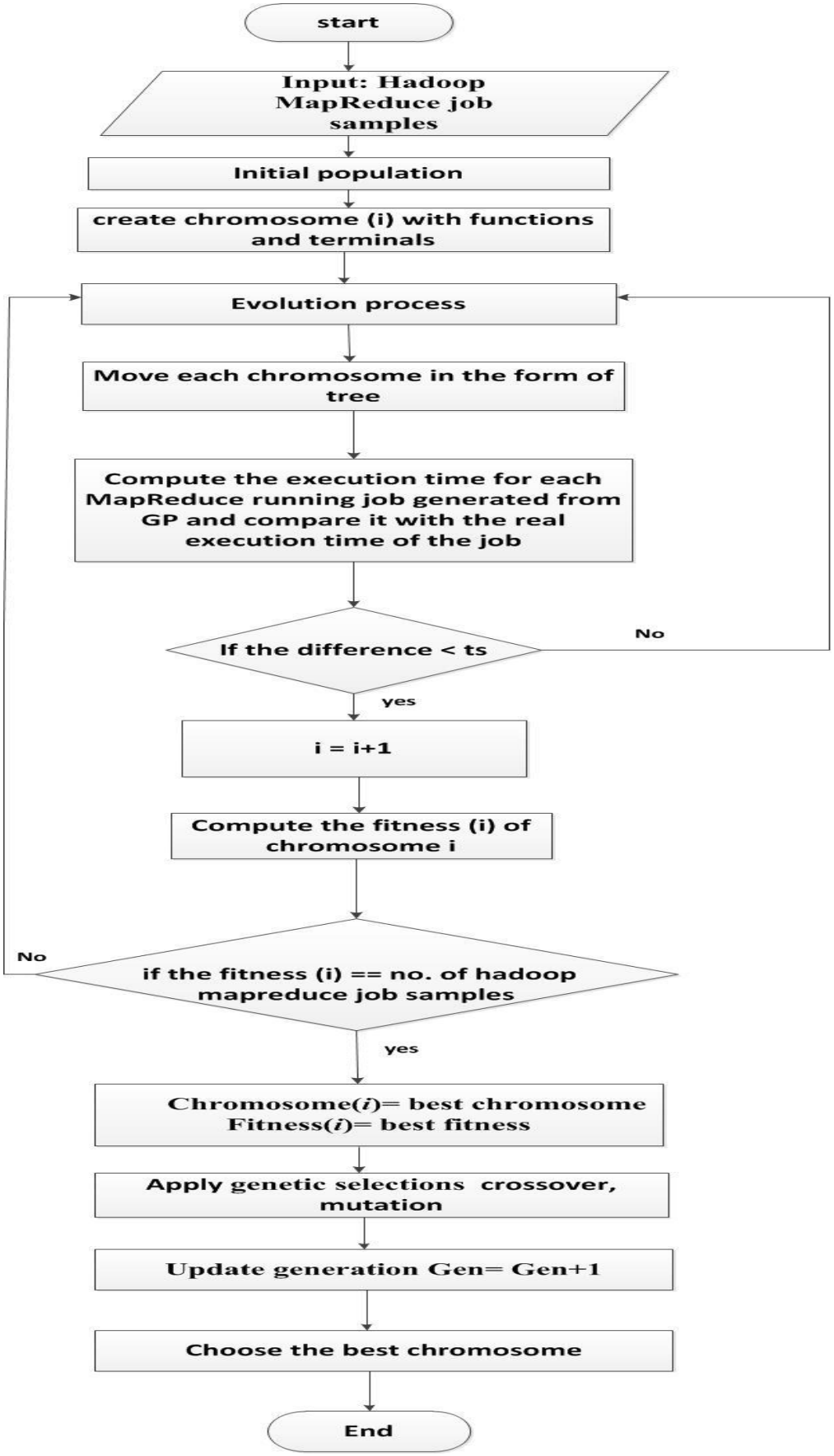

Figure 3. 2 The flowchart of genetic programming

In this work, a list of MapReduce jobs is used as input datasets and a large number of experiments was run for both Word count and Tera sort applications, being used to process different sizes of these input datasets, as presented in section 3.6.1. The implementation of GP is performed to find all possible expressions between the Hadoop MapReduce parameters by generating hundreds of chromosomes and in this work, 600 were initially generated. All chromosomes are represented in the form of tree graph and the fitness value of each is calculated based on the completion time of a Hadoop MapReduce job for each training dataset. The completion time is considered as an objective function of the genetic programming.

The completion time of a Hadoop MapReduce job f ($k1, k2,.., kn$) for training datasets generated from genetic chromosomes is compared with the real completion time of the Hadoop MapReduce job. The difference between the approximated and real completion time of the Hadoop MapReduce job should not be more than 40s, which is referred as TS. The chromosome with the high fitness value is selected. The measure of fitness value is the same as the number of Hadoop MapReduce job used in this process. This measure is supposed based on the example of soccer player to test the fitness in [100]. The evolution process will terminate once the best fitness value is obtained, i.e. when reaches to the number of Hadoop MapReduce jobs used in the process. Moreover, genetic selections and operators are applied, such as mutation and crossover, to produce new chromosomes and update the current ones. The expression between the parameters is obtained after 40,000 iterations. The probability of crossover and mutation was 0.9 was 0.05, respectively. Equation 3.1 represents the mathematical expression and the relation between the Hadoop MapReduce parameters, which is used as an objective function in the next algorithm (GA). The tree in Figure 3.3 illustrates the objective function. In this tree, eight parameters are taken as inputs. In the first part, the addition function is first applied to terminals k_3 and k_7. Next, the division is applied for terminals k_5 and k_2. The multiplication sign is used for the above-mentioned functions to generate intermediate result in the first part. In the second part, both the multiplication and the addition are used for terminals $k1$, k_6, k_4 and k_8, respectively to generate the intermediate results of the second part. Finally, the addition sign is applied for both parts to generate the final results.

$$f(k_1,k_2,\ldots,k_8) = (k_3+k_7)*(k_5/k_2)+(k_1*k_6)-(k_4+k_8) \quad (3.1)$$

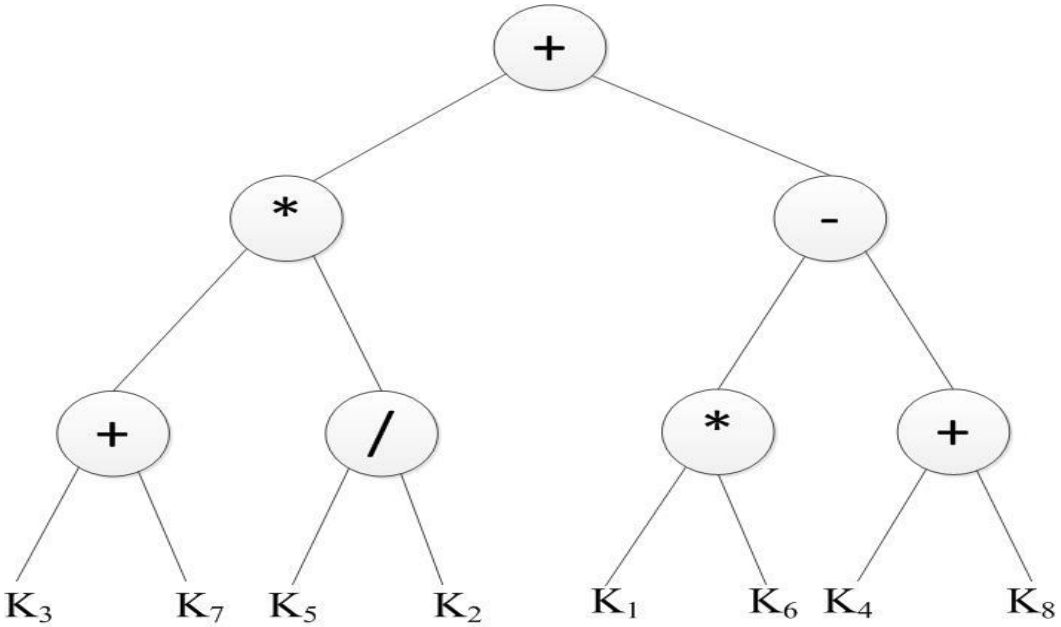

Figure 3. 3 the tree of the objective function

3.5 Hadoop MapReduce Parameter Settings Tuning Using a Genetic Algorithm

A genetic algorithm (GA) is a metaheuristic one, which belongs to the group of evolutionary algorithms (EA) and was first proposed by John Holland to provide better solutions to complex problems. GAs are widely used to solve many optimisation problems based on natural evolution processes. They work with a set of artificial chromosomes that represent possible solutions to a particular problem. Each chromosome has a fitness value that evaluates its quality as a good solution to the given problem [101]. GAs start with generating a random population of chromosomes. A set of essential genetic operations, such as crossover, mutation and update are applied on the chromosome to perform recombination and selection processes on solutions for specific problem. The selection process of chromosomes is performed based on their fitness value. The chromosome with high fitness has the chance to be chosen and create an offspring to generate the next population [102]. Figure 3.4 describes the procedure for GA implementation, where Equation 3.1 generated from GP is used as an objective function that needs to be minimised, which is expressed as:

$$f(k_1, k_2, .., k_n) = (k_3+k_7)*(k_5/k_2)+(k_1*k_6)-(k_4+k_8)$$

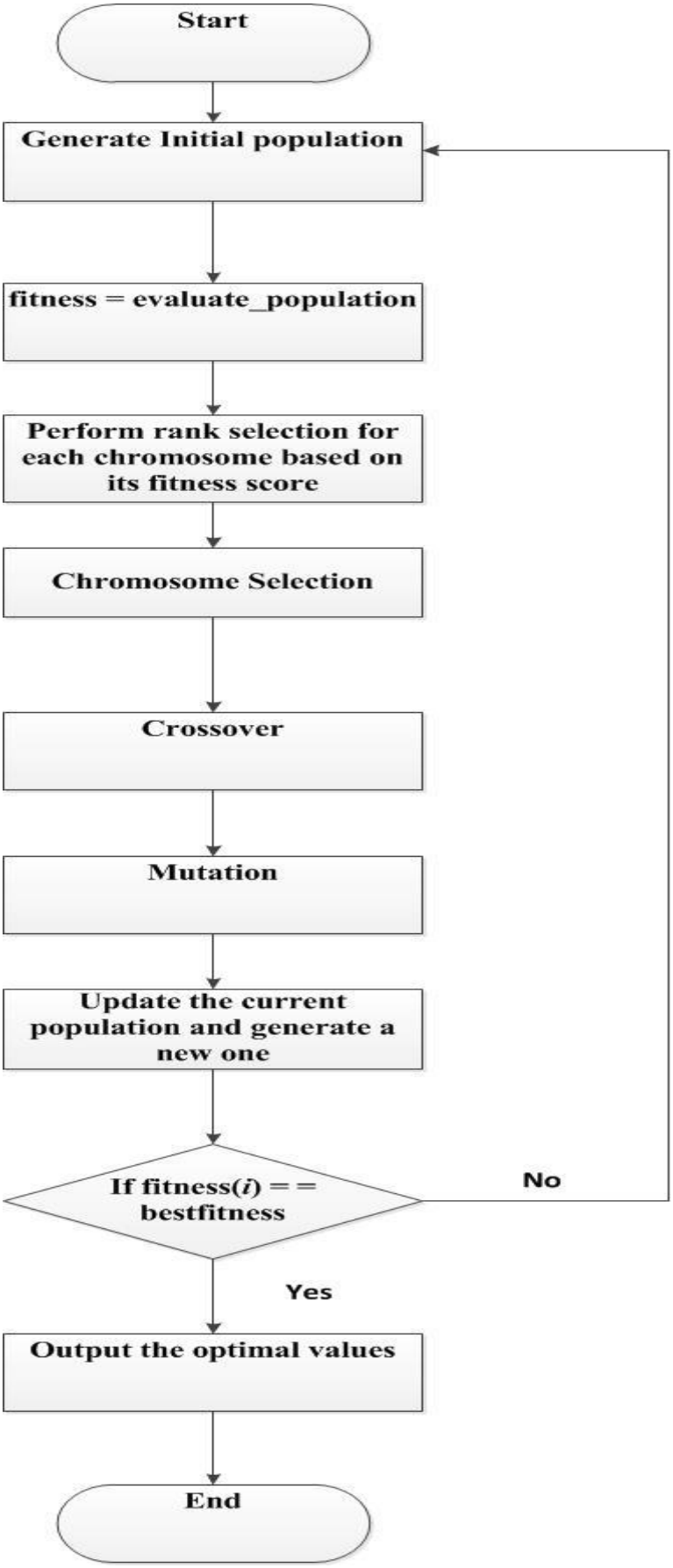

Figure 3. 4 The flowchart of the genetic algorithm

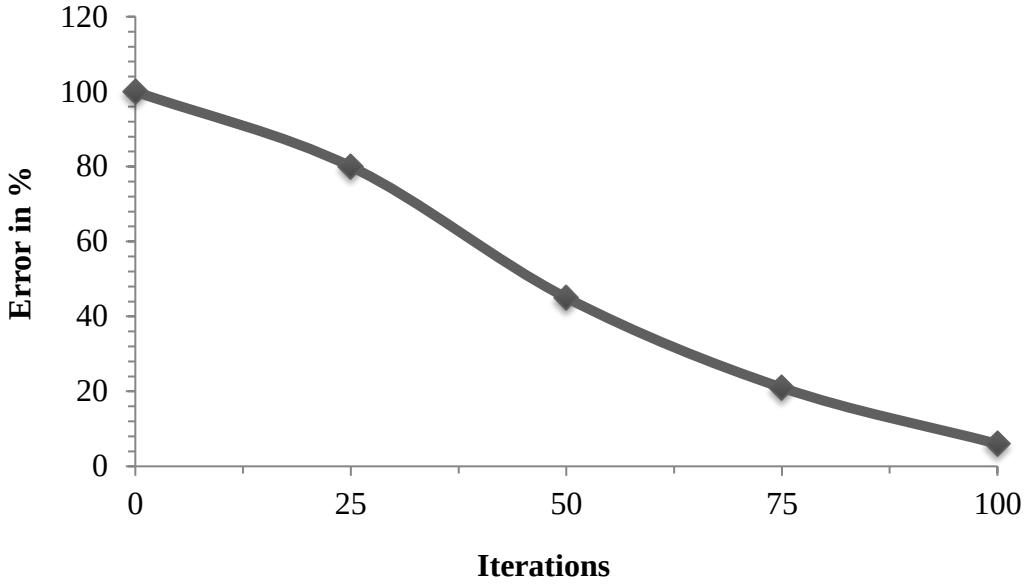

Figure 3. 5 The error against the iterations in GA

In algorithm 2, an initial population of chromosomes is randomly generated and each MapReduce parameter is represented as one of these. It means that chromosome (i) = $k_1, k_2,.., k_n$, where n is the number of parameters. As aforementioned, in this work, there are eight parameters that need to be tuned. After the generation of the population, the fitness value of each chromosome in it is evaluated based on the objective function $f(k_1, k_2,.., k_n)$. The chromosome with high fitness is selected and genetic operators, which are selection, crossover and mutation, are applied to update the current population and generate a new one. The procedures are repeated until the best fitness values of chromosomes, which represent the optimised MapReduce parameters, are obtained or the number of iterations is finished. In this algorithm, the population size is 15 and the number of iterations set to be 100. Furthermore, the probability of crossover Pc = 0.8 and the probability of mutation Pm = 0.05 are empirically determined and used as genetic operators. Roulette wheel spinning is employed as a selection process. The ranges and recommended values of the Hadoop MapReduce parameters in GA are presented in Table 3.2. The search space is performed based on these ranges provided in the Table 3.2. Figure 3.5 shows the performance of GA during the search of the optimal values of the Hadoop parameters. The performance is evaluated based on the error occurred during the search of the optimal values against the generation number. As mentioned before, the objective function is constructed based on the estimated execution time of Hadoop MapReduce job samples compared to real execution time.

The error should not be exceeding 15 %, considering the execution time of a Hadoop MapReduce job sample into account.

Table 3. 2: Hadoop MapReduce parameters settings in genetic algorithm

Hadoop MapReduce parameters	Range	Parameters name	Illustrations
K_1	100-165	MapReduce.task.io.sort.mb	Depends on the block size of an input data set. 64 MB is used as a block size
K_2	10-160	MapReduce.task.io.sort.factor	Empirically
K_3	True	Mapred.compress.map.output	
K_4	1-16	MapReduce.job.reduces	Depends on the total number of reduce slots used in the cluster
K_5	0.60-0.80	MapReduce.task.io.sort.spill.percent	Empirically
K_6	2-4	MapReduce.tasktracker.map.tasks.maximum	Depends on the specification of a data node (slave node)
K_7	2-4	MapReduce.tasktracker.reduce.tasks.maximum	Depends on the specification of a data node (slave node)
K_8	0.70-0.71	MapReduce.reduce.shuffle.input.buffer.percent	Empirically

3.6 Performance Evaluation Environment

The proposed work was implemented and evaluated using eight virtual machines (VMs) of a Hadoop cluster placed on Microsoft azure cloud. Each VM was assigned with 8 GB memory, 4 CPU cores and 320 GB storage for the whole cluster. Hadoop Cloudera

(Hadoop 2.6.0-cdh5.9.0) was installed on all nodes, with one being configured as a master and the rest as slaves. The master node could also be run as a slave. For fault-tolerance purposes, we set the replication factor of the data block at 3 and the HDFS block size was 128 MB. Table 3.3 presents the specifications of Hadoop cluster.

Table 3. 3: Hadoop cluster setup

Intel Xeon X5550 server 1 and uxisvm04 server 2	CPU	4 cores for each VM
	Processor	2.27 GHz
	Hard disk	360 GB
	Connectivity	1 GBit Ethernet LAN interconnectivity between two servers
	memory	64 GB
Operating System	Host Operating System	Microsoft windows server 2012 R2
	Guest Operating System	Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

3.6.1 Experimental Results

Both the Word Count and Tera sort applications have been run as real job programs for Hadoop MapReduce framework to evaluate the performance of our proposed work on a Hadoop cluster. It can be clearly observed that there is a difference among the tuned configurations of the Hadoop MapReduce parameter settings using our proposed system, the default one and Gunther's method. For instance, Figure 3.6 shows that when the value of *io.sort.mb* increases, this leads to a decrease in the execution time of the Hadoop MapReduce job. The reason for this is that increase the value of this

parameter results in less operation during of spill records to the hard disk. Moreover, the *io-sort-factor* parameter defines the number of data streams to merge during the sorting of files. From Figure 3.7, it can be clearly seen that when the value of this parameter goes up, the execution time of the job goes down. The reason is because increase the value of this parameter leads to less overhead during IO operations.

It can also be observed from Figure 3.8 that when the number of reduce tasks is increased from 5 to 10, the execution time of the Hadoop MapReduce job decreases. However, increasing the number of reduce task results in longer execution time due to the overhead of network resources as well as over utilisation of computing resources, such as CPU and memory. Moreover, it is evident that any further increase in reduce tasks leads to the generation of high network traffic and consequently, an increase the overall time of the Hadoop job. Figure 3.9 shows that increase in the slots of map and reduce can play crucial role for better utilisation of cluster resources and accordingly minimise the overall time. One slot has been configured per CPU core, in the cluster setup 4 cores has been allocated for each cluster node and therefore 4 slots has been employed to maximise the utilisation of CPU. If additional slots are included in the setup, this exhausts the CPU and results in a delay in the processing time of the MapReduce job.

Figure 3.10 shows the completion time of MapReduce jobs for different sizes of datasets by applying a compression parameter. It is observed that applying this parameter can reduce the completion time of a MapReduce job by alleviating the traffic consumption of the network and reducing the pressure on the I/O operation. However, the compression of input data and reduce output data is not available in some applications such as Tera sort. Moreover, the performance of this parameter is reduced when massive datasets are used such as, 40 or 50 GB. The reason for this is that any increase in dataset size leads to the generation of high volumes of shuffling traffic, especially in a static IP network environment. As a result, a software defined network is implemented on a Hadoop cluster to reduce the shuffling traffic generated from a MapReduce job. The following section describes the implementation of a Hadoop cluster based on SDN.

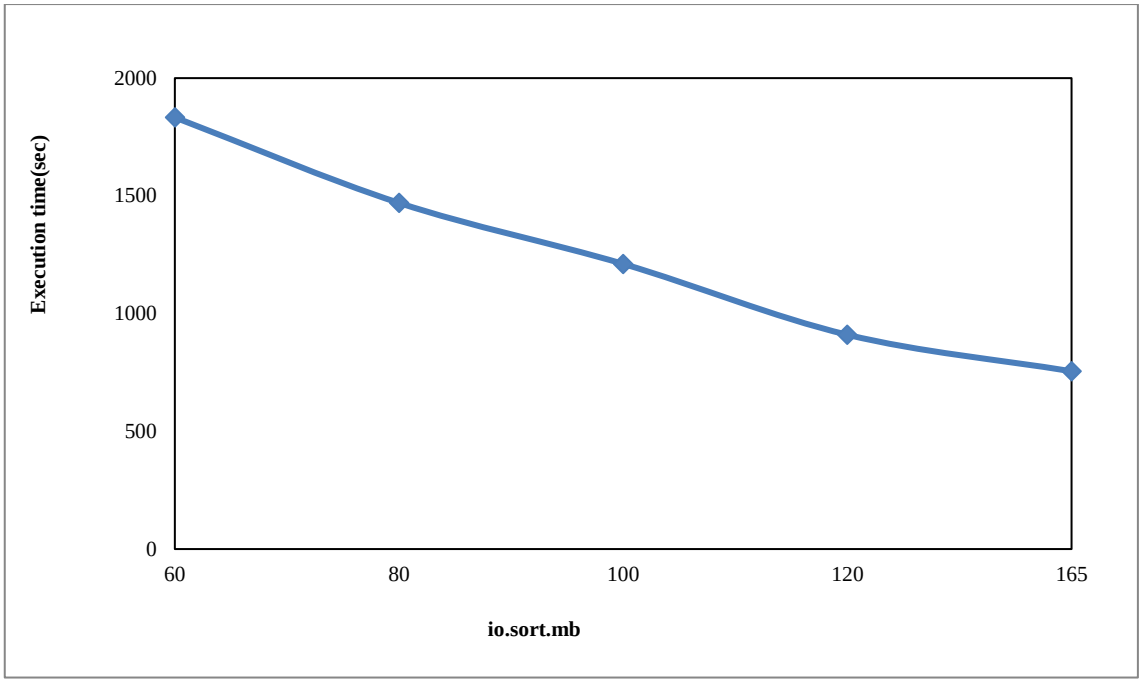

Figure 3. 6: The effect of the *io.sort.mb* parameter (K_1)

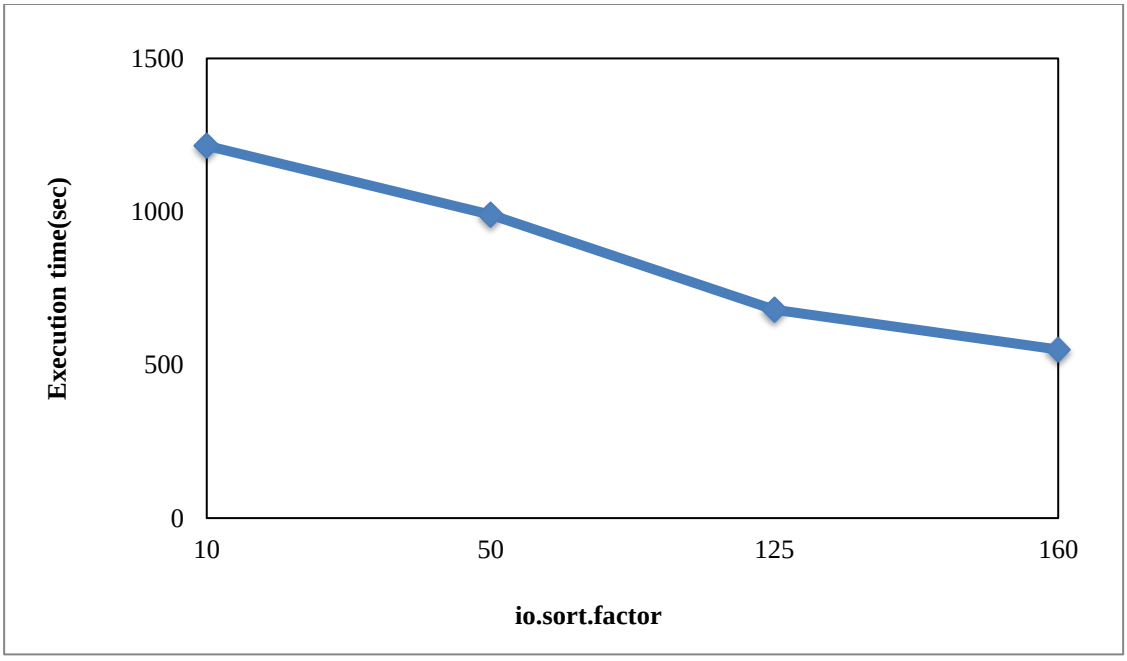

Figure 3. 7: The effect of *io.sort.factor* (K_2)

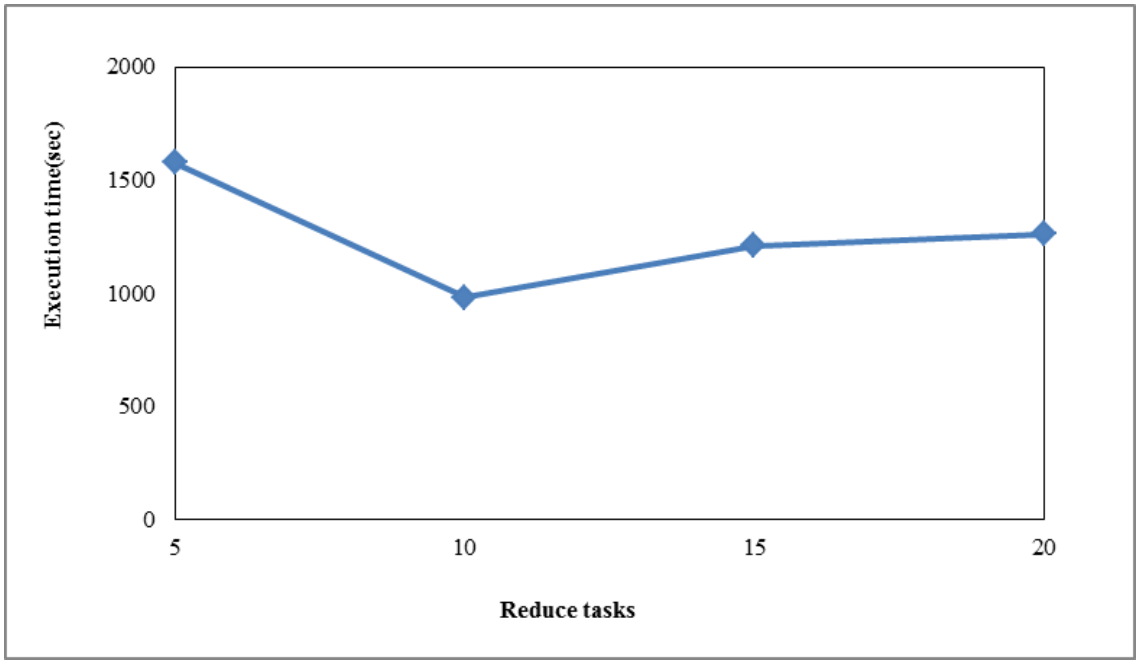

Figure 3. 8: Reduce tasks influence (K_4)

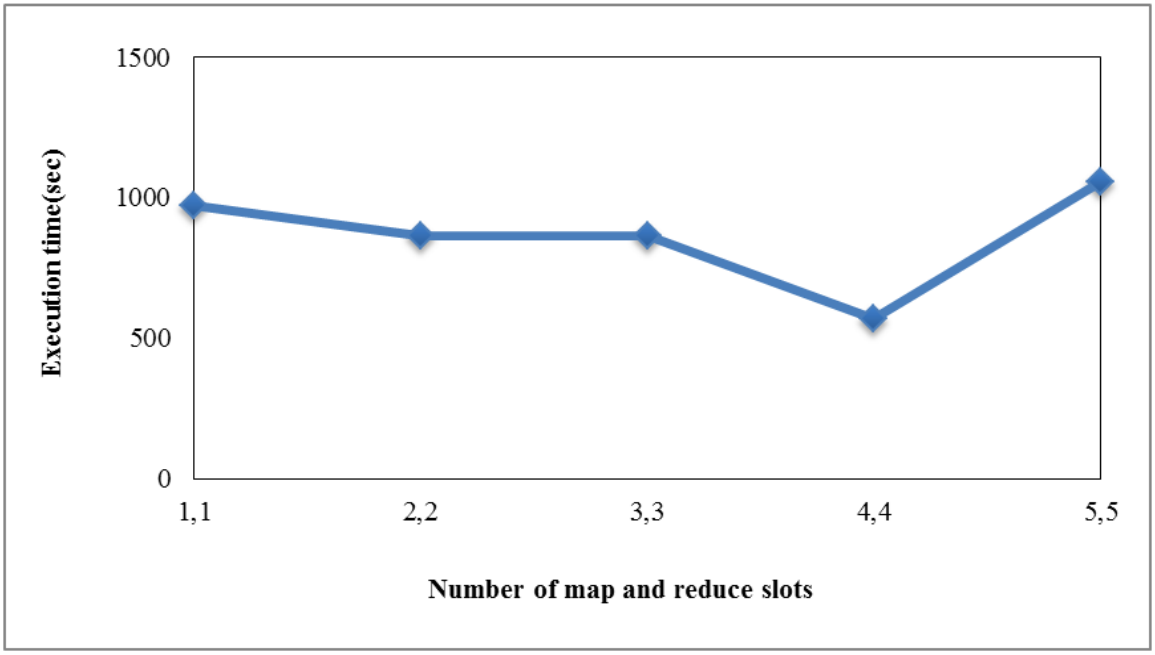

Figure 3. 9: Map and Reduce slots influence on MapReduce job (K_6 & K_7)

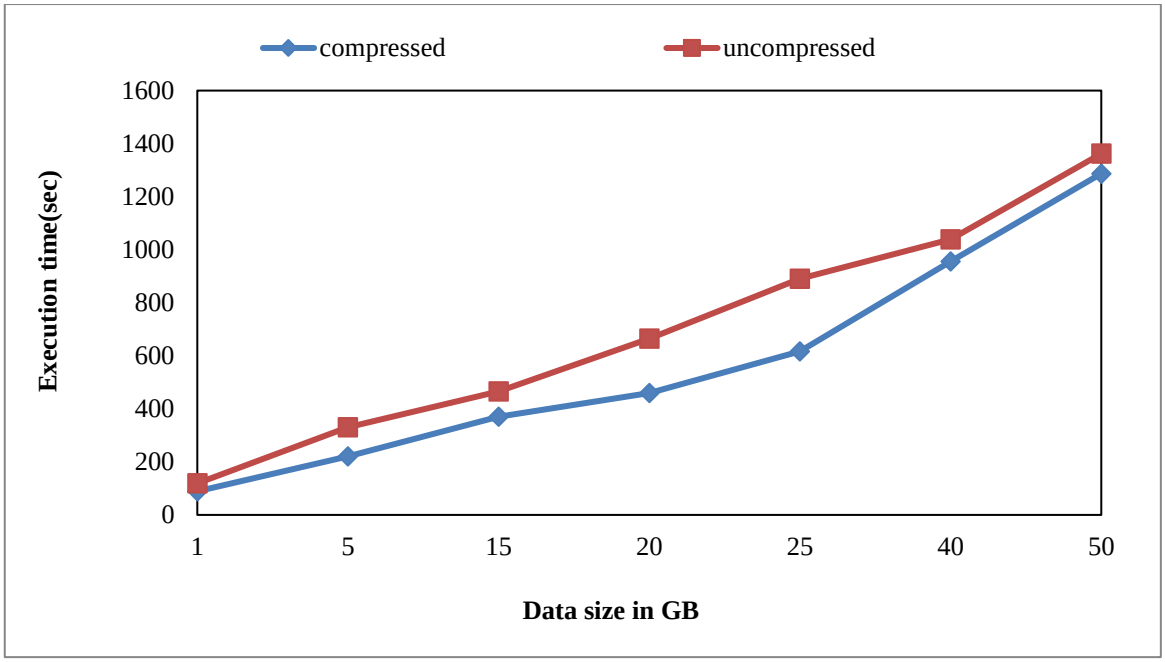

Figure 3. 10: compression parameter influence (K_3)

Table 3. 4: Hadoop MapReduce parameter settings recommended from genetic algorithm on eight virtual machines

Name	Default	Optimised Values using Genetic algorithms		
		1GB	10GB	20GB
mapreduce.task.io.sort.mb	100	100	140	165
mapreduce.task.io.sort. factor	10	50	125	160
mapred.compress.map. output	false	True	True	True
mapreduce.job.reduces	1	16	10	10
mapreduce.map.sort.spill.percent	0.80	0.87	0.68	0.77
mapreduce.tasktracker.map .tasks.maximum	2	4	4	4
mapreduce.tasktracker. reduce.tasks.maximum	2	4	3	4
mapreduce.reduce.shuffle .input.buffer.percent	0.70	0.70	0.71	0.71

Table 3.4 shows the optimised values of the Hadoop MapReduce parameters for each size of dataset on eight virtual machines. It is worth mentioning that the hardware resources in terms of CPU and memory as well as the size of an input data set are considered by the implemented GA algorithm. The size of input datasets should be considered before setting specific values for each parameter. For instance, setting a large number of reduce tasks for small data sizes like 1 GB better utilises hard disk during the parallelisation of tasks. However, high overhead is generated when setting such large number of reduce tasks. Therefore, small number of reduce tasks that can be finished in a single wave should be configured for small data sizes like 1 GB to reduce the overhead during the configuration of the reduce tasks. To show the performance of our method, different sizes of data, including 1 GB, 10 GB and 20 GB, were generated. The tuned parameters were used for both the Word Count and Tera sort applications. The execution time of both the word count and Tera sort applications based on the tuned settings by our proposed method is compared with the execution time of the two applications based on the default setting as well as the settings achieved by Gunther. Both Word count and Tera sort were run twice and it emerged that our proposed method can improve the performance of a MapReduce job in a Hadoop cluster, most notably with large input data sizes.

Figure 3.11 and Figure 3.12 show the completion time of a Hadoop MapReduce job using the proposed method in comparison with the default one and Gunther's method. From Figure 3.11, it can be observed that the performance of the Hadoop Word Count Application is improved using the proposed approach by 63.15% and 51.16% for the 1 GB dataset when compared with the default and Gunther's settings, respectively. Furthermore, the experiments carried out on a 10 GB dataset show that our proposed method improves the performance of the Word Count Application by 69% and 37.93% when compared with the default and Gunther's method, respectively. Finally, the proposed method also achieved better performance than the default and Gunther settings on the Word Count application by 69.62% and 30.31%, respectively, for 20 GB.

From Figure 3.12, it can be clearly seen that our proposed method improved the Tera Sort application performance by 52.72% over the default system and 44.28% when compared to the Gunther settings for 1GB. For 10 GB, the performance was improved by 55.17% as compared to the default one and was 51.25% better than with Gunther's

method. Finally, Tera Sort application performance for 20 GB was improved by 73.39 % and 55.93 % more than the default and Gunther settings, respectively.

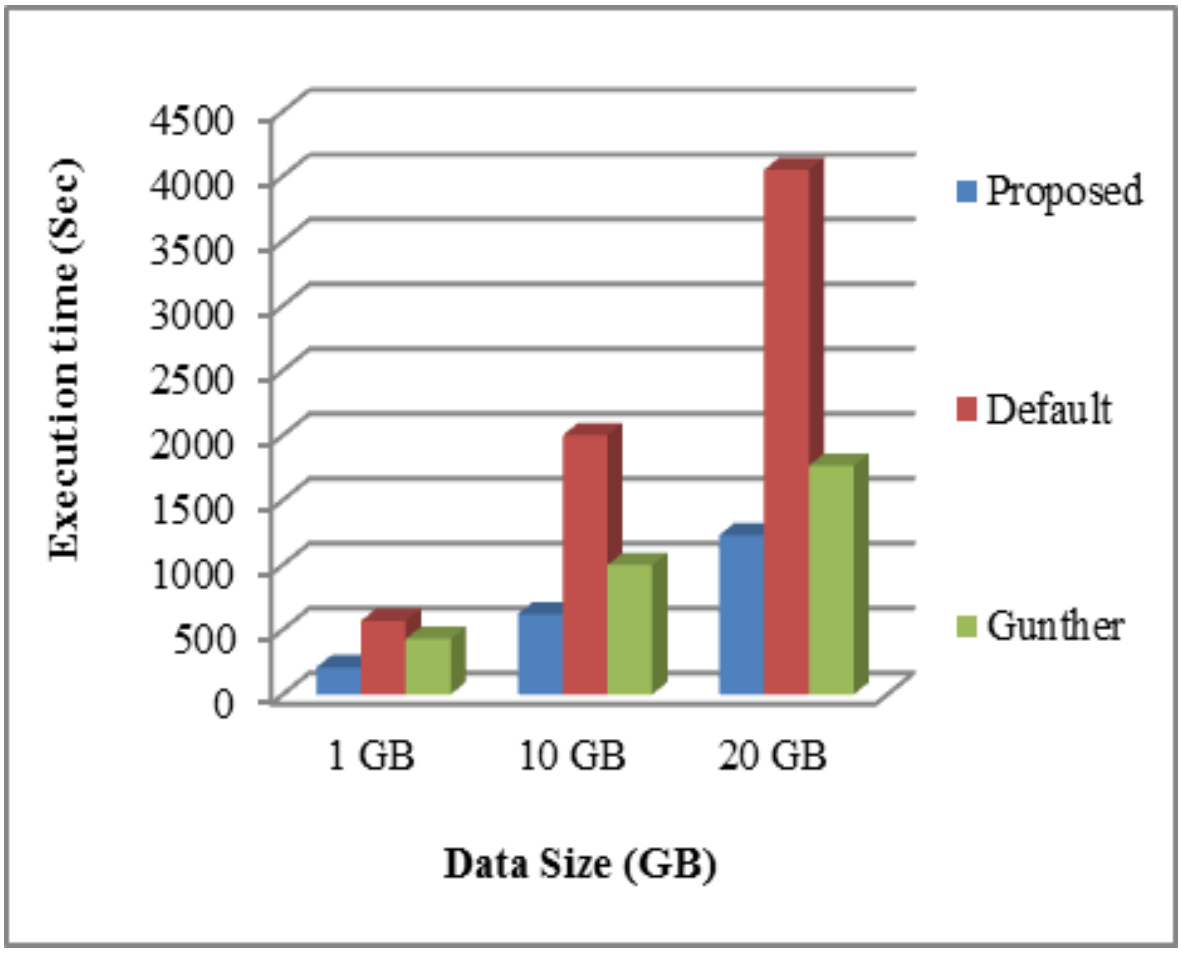

Figure 3. 11: Comparison of Word Count Application

Figure 3. 12: comparison of Tera sort application

3.7 Hadoop Cluster Based on Software Defined Network

Software defined network [103] is an emerging technology that provides agile and dynamic management for the network through central and intelligent programming. In this novel technology, the control plane is decoupled from the data plane to provide more flexibility and agility, which leads to network performance improvement by obtaining better routing decisions. The controller communicates with the OpenFlow switch through the OpenFlow protocol. In this work, SDN is implemented to improve the performance of Hadoop networking by efficient utilisation of bandwidth for shuffling traffic. Different sorts of traffic are generated from a Hadoop cluster, such as shuffle phase traffic, HDFS data transfer, HDFS read and write along with Hadoop monitoring messages. It is worth noting that the shuffling traffic represents the most traffic produced by both Word Count and Tera Sort in a Hadoop cluster followed by HDFS read and write. In the proposed system, SDN is employed with openVswitch to allocate more bandwidth for the traffic generated by the shuffling phase when the mapper transfers its output to the reducer. However, identifying the network resources of shuffling traffic is a challenging task, because the core framework of Hadoop does

not include sufficient information regarding network resources demand for this traffic. A Hadoop cluster has a single job tracker and several task trackers. The progress of Hadoop jobs is monitored by the job tracker, whilst each task tracker sends heartbeat messages to the job tracker about its status. However, these messages lack sufficient information about the network resources. To address this, our proposed system installs software engines on each Hadoop host to record the required information of network resources for each shuffling flow. This information contains the size of map output data (intermediate data) being transferred over each flow to the reducers.

Furthermore, software engines determine the required network bandwidth for each shuffling flow and record sufficient information, such as the IP address of the source and destination nodes as well as the size of each flow. Then, all the required information is delivered to the SDN controller to assign an efficient bandwidth for shuffling flows. The SDN controller installs flow entry in each Open vSwitch for each shuffling flow and moves the shuffling flows to a queue with higher bandwidth. On the other hand, flow rules are installed in Open vSwitch for other types of traffic, such as control messages and HDFS read/write, to switch them to another queue with low bandwidth allocation. The TCP communication between the task trackers to send the map output data in a Hadoop cluster is performed using port 50060. Open vSwitch matches the incoming packets to identify them by their port number.

In the proposed system, 14 virtual machines, installed on two servers, were used with two packages of Open vSwitch installed on two PCs, with one floodlight SDN controller being installed on one PC. SDN application was installed on another PC to install flow entries on each open vSwitch and to manage and administrate MapReduce jobs in a Hadoop cluster as well as the replication of Hadoop distributed file system (HDFS) blocks through web interface. The two servers were connected to a single physical switch with 1GB link capacity.

Figure 3.13 shows the proposed cluster based on an SDN environment. Both Word Count and Tera Sort applications were used to evaluate our proposed system using SDN technology. The experimental results show that our proposed system based on an SDN environment improves the performance of the Word Count application by reducing the completion time up to 12.4% for 30 GB when compared to a TCP/IP environment. Moreover, this rises to 21.9% for 40 GB, while for 50 GB, the completion time is

reduced by 32.8% when compared to a TCP/IP Hadoop cluster, as shown in Figure 3.14. Figure 3.15 shows the performance for the Tera Sort application using the proposed system for different data sizes ranging from 30-50 GB. It emerges that the proposed system reduces the completion time of Tera Sort for 30 GB on average by 53%. Furthermore, the completion time for 40 GB and 50 GB is reduced by 48.1 % and 38.7%, respectively, over a TCP/IP environment. It is worth noting that performance of Tera Sort application decreases with larger data sizes due to the high volume of shuffling traffic that is generated form these jobs.

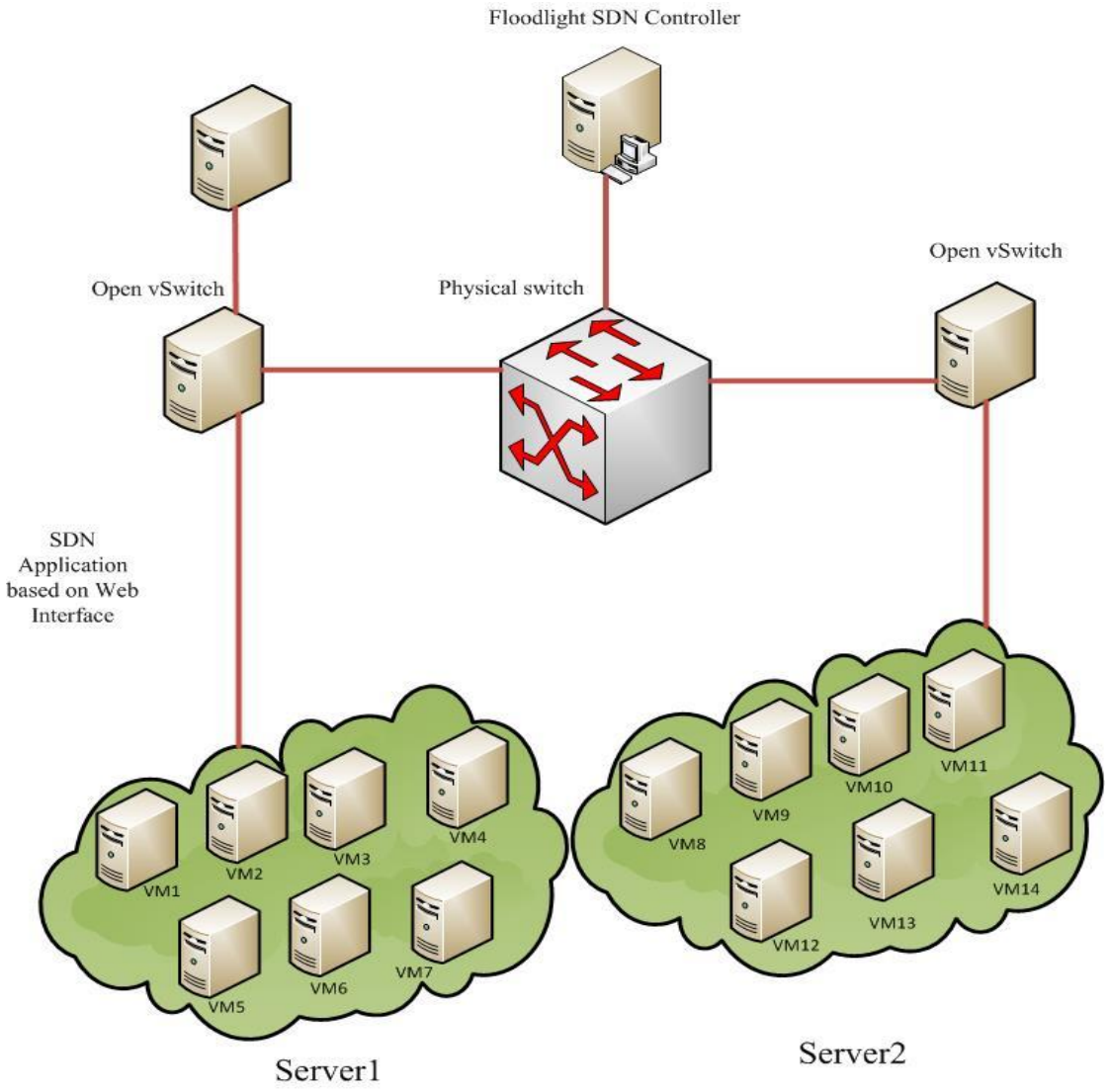

Figure 3. 13: Small scale Hadoop cluster in SDN environment

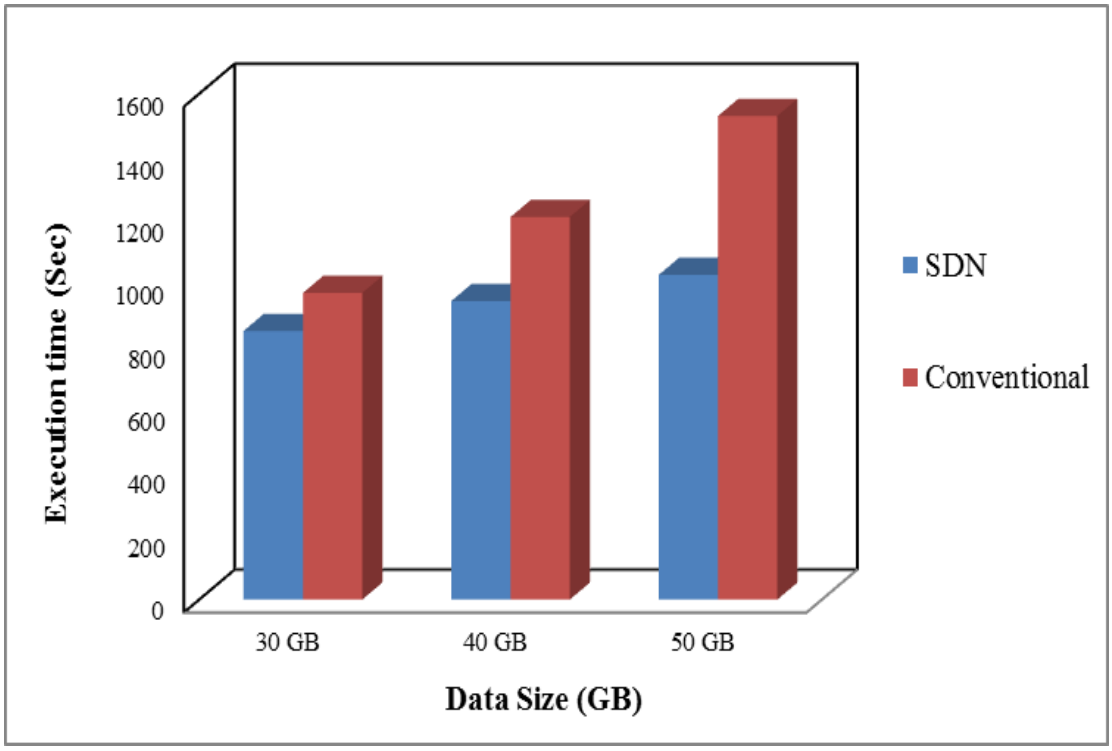

Figure 3. 14: Word Count Performance in SDN Environment

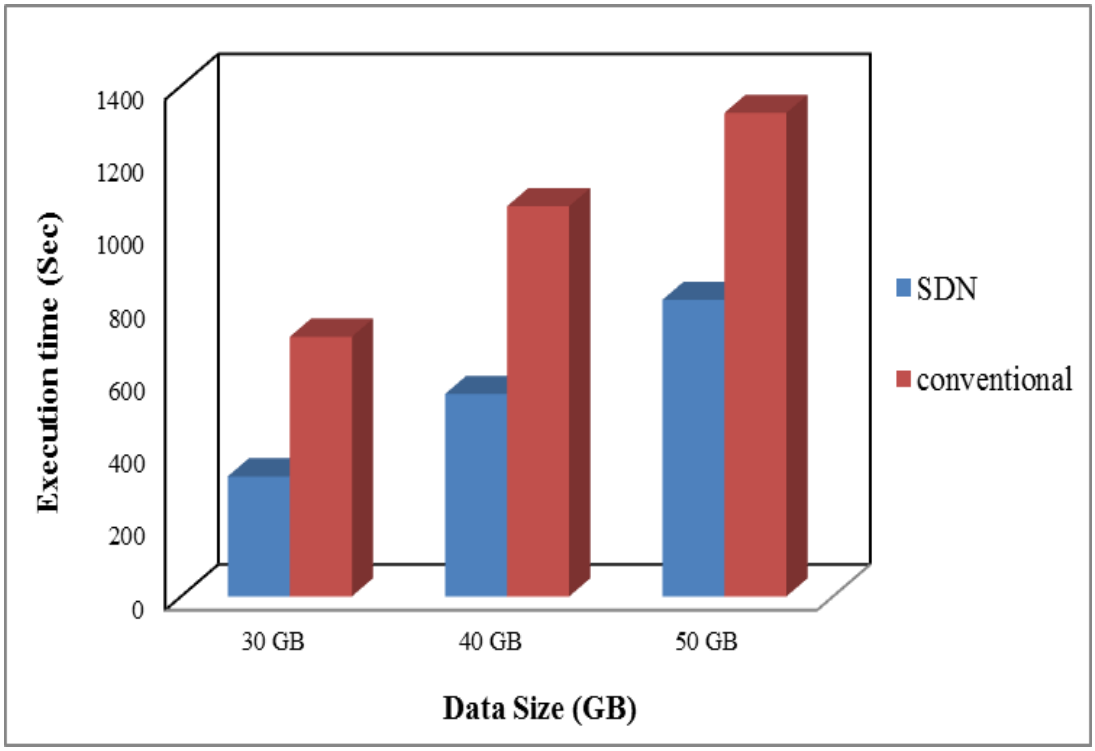

Figure 3. 15: Tera sort Performance in SDN Environment

3.8 Summary

In this chapter, both of Genetic Algorithm and Genetic Programming have been used to tune the configuration parameters of Hadoop MapReduce automatically. By optimising the configuration parameter settings, the computing performance of a Hadoop framework can be improved and then reduces the completion time of Hadoop MapReduce jobs. Further optimisation has been performed using software defined network technology. Two applications, which are Word Count and Tera Sort, have been run to evaluate the MapReduce job performance of the Hadoop framework. This work was evaluated using a cluster consisting of eight virtual machines placed on internal cloud in Brunel University London. Another cluster of 14 virtual nodes was employed based on SDN. The experimental results in traditional network using 8 VMs showed that our proposed method improved the MapReduce job performance in a Hadoop cluster in comparison with Gunther and default settings. Moreover, the results using 14 VMs based on SDN network showed that the performance of a Hadoop job has been further improved when compared to traditional network.

Chapter 4

Solar Radiation Time Series Analytics Based on R and Hadoop Integrated Programming Environment

Solar energy has become the most important source of renewable energy. However, it is mainly dependent on the weather conditions and solar irradiation. The calculation of daily solar irradiation is very important to estimate the final output power of solar arrays. There are some computational and storage challenges for the calculation of the solar irradiation collected by massive solar panel arrays. It is a challenging task to process and store massive solar irradiation data by the current calculation tools like R language or solaR. Effective technique is highly required to accelerate the calculation of the solar irradiation data of large solar arrays. This chapter presents an effective technique for fast statistical calculation for huge solar irradiation data by using R and Hadoop integration programming language environment (RHIPE). It is evaluated using data provided by the London Weather Centre for the period (1996-2005) that includes monthly mean daily irradiation on inclined planes in terms of scalability, speedup and accuracy. The speedup of RHIPE in computation is first analysed through Gustafson's law. This law is revised to improve its ability to analyse the performance achievement in computation for parallelising data in a cluster computing environment like Hadoop.

4.1 Introduction

With the emergence of Internet of things (IoTs), huge data are generated every day. Different types of data are produced from various sources including social media, sensors, financial sector and healthcare. Time series data is one of the main sources of generated data. However, it has big challenges in terms of processing and analysing. Moreover, another challenge is the deriving of knowledge from time series data that can help to improve the productivity [104]. Time series is any data gathered sequentially in time such as a series of data points recorded over a continuous time interval. The time series data arises from different sources including Economics and Finance, environmental modelling, meteorology and hydrology, demographics, medicine, engineering and quality control [105] . One of the most important sources of time series data is the solar data, which includes the temperature and irradiation produced by sun.

One of the best tools to analyse all types of time series data including solar data is R language. R is an open source environment to perform statistical analysis and data visualization. Many data scientists and statisticians used R language to perform statistical analysis and discover key insights from data using techniques such as regression, classification, clustering, recommendation and text mining [61]. However, due to the large time series data sets produced from many sectors, it has become difficult to store, analyse and process these big time series data sets. As a consequence, Hadoop MapReduce Framework can be integrated with R language to perform the analysis of time series data. Hadoop MapReduce is robust computing platform used for big data analytics. Hadoop is an open source framework implementation of MapReduce algorithm. It has superb features such as high scalability, fault-tolerance and data parallelization. It is also able to distribute data and parallelizes computation through a cluster of computing nodes. Therefore, parallel computing is an effective way to improve the performance in terms of accuracy and timely analysis of solar time series data. In this chapter, the design and implementation of integration of Hadoop and R language for fast calculation and analysis on massive amounts of solar time series data is presented.

4.2 Overview of Big Data Analytics Based on Cloud Computing and High Performance Computing

Nowadays, many individuals and companies use IT and communication technologies to develop their own business. Consequently, massive data sets are enormously generated leading to some computational issues. The traditional tools face some challenges in terms of computation and storage. For example, it has become a challenging task for the traditional tools to store and process huge amount of data sets that generated from different sources at a high pace. These challenges can be solved through the implementation of the parallel computing and processing by using the technology of high performance computing (HPC).The computational problems can be tackled using multi-processor system ranging from 2 to 64 CPUs. The multiprocessor system consists of number of High performance RISC (Reduced instruction set computer) processors. Furthermore, the multiprocessor system is designed to access the same memory through symmetric multi-processing (SMP). However, applying multi-processors to deal with a

single problem requires an efficient programming to achieve an effective processing. Moreover, this system is designed to solve small applications only.

As a result, it has become necessary to provide a reliable solution to fit large applications. The solution can be found in the use of multi- computing nodes through using a message passing tool such as parallel virtual machine (PVM) or message-passing interface programming (MPI) to achieve parallel computing over the large problems and applications. Another system called NUMA can be used to solve large problems through a number of interconnected processors that can access a distributed shared memory using a load/store paradigm similar to SMP system [106]. Cloud computing can be a valuable remedy for processing and storing large applications that do not fit into memory and cannot be processed using HPC. Cloud computing can provide some computing services for processing big data applications by using amazon elastic MapReduce (Amazon EMR). Amazon EMR uses a cluster of computing nodes that run on virtual servers provided by Amazon Elastic Compute Cloud (EC2). Apache Hadoop is installed on each node in the cluster. Apache Hadoop is a framework that currently deployed to provide distributed storage and parallel processing for data intensive applications. It distributes the computation and parallelises the execution of big data jobs across several machines in a cluster computing environment. Hadoop uses MapReduce programming to implement the parallel processing by writing two functions (map and reduce). More details on the MapReduce programming model and the Hadoop framework are presented in section 4.3 and 4.4, respectively.

With the increasing demand of the renewable energy such as wind and solar, the electric power is becoming more complicated from the aspect of secure storage and computation process. For instance, solar energy has attracted many researchers to study the characteristics and behaviour of its resources in depth. One of the most important aspects of solar energy is its unpredictability. It is becoming very important to have a balance between the demand of power and generation. Consequently, it has become very necessary to estimate how much power can be produced using solar arrays. The power produced by solar arrays mainly depends on the availability of the solar irradiation and since the solar irradiation is unpredictable and relies on the weather conditions. The solar radiation that comes from the sun is a number of values that can be calculated on daily, weekly or monthly basis and these values are considered as a time series data. Nowadays, large number of solar arrays being installed in different

locations around the world. This numerous number of solar arrays will produce huge amount of time series data such as the irradiation that comes from the sun. As a result, huge amount of data will be generated and it is very complex to analyse and store these large datasets based on traditional methods.

High performance methods can be used to analyse and store theses datasets through parallel processing. For instance, massage passing interface (MPI) is a parallel programming model that used to distribute and parallelize computation across number of processors or computing nodes. In [107] MPI system model has been employed to parallelize computation jobs across several nodes of grid computing. High Performance Computing environment (HPC) was used by [108] for parallel contingency analysis. However, MPI has some weaknesses in some parts such as topology awareness, fault tolerance and scalability [109]. Cluster computing can be used as an alternative approach. In [110] a high performance hybrid computing approach was used for massive contingency analysis in the power grid. A XMT multithread C/C++ compiler on Gray XMT (multithread HPC computing platform) and conventional Cluster computers were employed to parallelize the algorithm. Parallel computing toolbox within MATLABs Distributed Computer Server (MDCS) was used to parallelize contingency analysis algorithm on multiple Processors [111]. In [112] a distributed framework for efficient analytics on ordered datasets using Hadoop is proposed. This framework adopts an efficient group-order–merge mechanism to speed up the execution of Re-Org tasks. In [113] runtime framework for automatic and transparent parallelization of the popular R language used in statistical computing is proposed.

A number of research works has been done on time series data. The work described in [114] proposes effective and distributed framework called R2Time to process data in Hadoop environment. This work has integrated R language with a distributed time series database (Open TSDB) using MapReduce programing technique (RHIPE) that supports analysts to work on massive datasets within robust analysis environment. Another research work proposed in [115] parallel genetic algorithm for the automatic generation of test units. This work is based on Hadoop MapReduce due to its high scalability and fault tolerance. Furthermore, this proposed work evaluated the speed-up with respect to the sequential execution. Gowri and Rathipriya [104] used the Map Reduce based genetic Algorithm for biclustering time Series Data. This work used the biclustering approach for time series data. For the optimal mining of patterns, genetic

algorithm was employed in this work. Moreover, MapReduce is also used to overcome the complexity of Big Data issues.

In [116] rolling win Novel framework was proposed for time series data prediction based on MapReduce. This proposed work used R programming language and Hadoop platform to support parallelization and fault tolerance. MapReduce algorithm was proposed with Ant Brood Clustering with Intelligent Ants) for clustering financial time series data in [117]. The work presented in [118] employed MapReduce framework and extended kalman filter based echo state network for time series prediction. The work proposed in [119] used distributed processing for time series data analysis based on pig and Hadoop cluster. A distributed parallel approach for anomaly detection was presented in [120]. MapReduce framework was used with this approach to detect automatic aberrant behaviour in large time series data sets. In [121] MapReduce was employed with dynamic time warping (DTW) for fast similarity search in time series data mining. The work described in [122] details an adaptive time series forecasting of energy consumption using optimized cluster analysis. Another number of research works has been conducted on the solar irradiation analysis. The proposed system in [123] developed a novel model to estimate solar irradiation on hourly and daily basis for hydrological studies. In [124] simple recurrent neural networks (SRNNs) was used to build a predictive model to predict the density of daily solar irradiation.

The work presented in [125] proposes a statistical model for solar radiation characterisation. In the proposed work, the daily variability of solar radiation is considered to produce synthetic time series from the measured and historical data of solar radiation. Four statistical methods were considered by the system proposed in [126] to estimate the global radiation data which is inaccessible for all locations. As a result, this system was developed to estimate the daily global radiation statistically based on the continuous series of other measurable meteorological parameters. Statistical analysis was conducted on the solar irradiation data to determine the error between the actinography values in relation to the pyrometer [127]. The proposed work was evaluated based on data from the Santa Maria University in Valparaiso that measured by the actinography and the pyrometer. SolaR is a statistical tool proposed by [128] and designed to calculate the daily and inter-daily global horizontal irradiation by using a set of classes and functions. This tool can also calculate the final output power of grid-connected PV systems.

4.3 MapReduce Programming Model

MapReduce is a parallel programming model for processing massive datasets across a cluster of computer devices in distributed computing environment [17]. MapReduce is a powerful computing technology for many intensive applications because of its characteristics such as fault-tolerance, scalability and elasticity. In MapReduce, thousands of computing nodes can be used to support parallel and distributed processing and analysis. With MapReduce model, computational tasks will be partitioned into Map and Reduce phases. Map phase divides the input data into several tasks to be performed in parallel across cluster of computers or virtual machines. Map task processes the input datasets to generate a list of intermediate key/value pairs. While the Reduce phase takes all intermediate values and merge them in a single key to generate the final result. Figure 4.1 shows the execution of MapReduce.

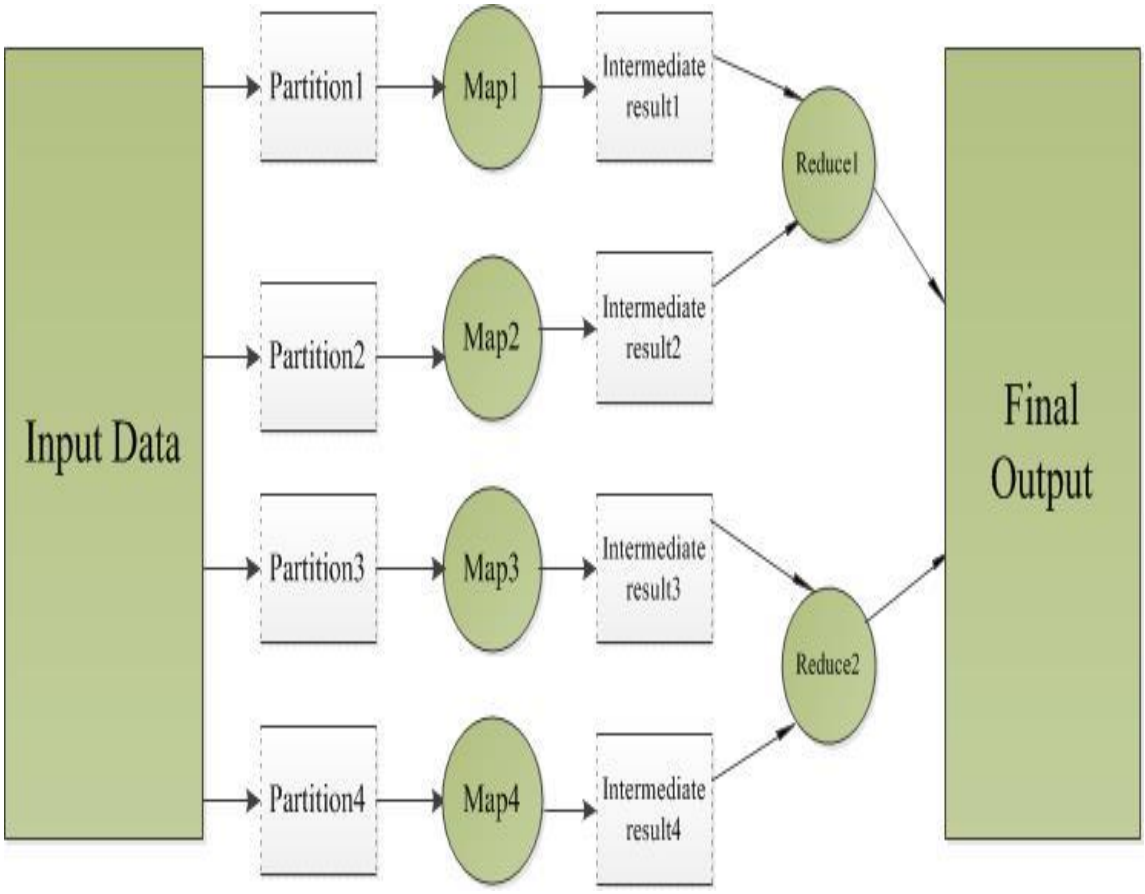

Figure 4. 1: MapReduce execution

4.3.1 The Implementation of MapReduce with Hadoop

The implementation of the MapReduce programming model can be performed using a number of systems such as Dryad [129], Mars [130], Phoenix [131] and Hadoop [132]. Hadoop is one of the most popular systems for MapReduce implementation because of its open source nature. The following section details more information about Hadoop framework.

4.4 Hadoop Framework

Hadoop is an open source framework written in java used for processing, analysing and querying massive amount of data across large clusters of commodity nodes. Hadoop is an apache project was originally developed by Yahoo and Doug Cutting [61]. Hadoop has useful characteristics which involves scalability, simplicity and fault-tolerance. It enables distributed processing of huge amount of data using MapReduce algorithm. Moreover, Hadoop can deal with both structured and unstructured data and can work on cluster computing environment and cloud computing system. Hadoop employs HDFS (Hadoop distributed file system) for data storage. HDFS is a distributed file system provides storage for huge amount of data across a massive number of computer clusters. HDFS is also able to provide rapid and scalable access to data [132]. All input data are stored on HDFS. Massive data is automatically divided into blocks that are managed by different nodes within Hadoop cluster. HDFS support master/slave architecture. It has single master node or Name node which acts as a server and multiple slave or data nodes which act as clients. File system namespace and access regulation to files by clients is managed by the master node. While the data nodes are responsible to manage storage attached to each data node and performs the execution of Map and Reduce tasks [7]. The name node has job tracker that divides jobs into multiple tasks and schedules the tasks on the data nodes. It also monitors and reassigns the task in case of any hardware or software failure. Data nodes have task tracker that is responsible to receive all Map and Reduce tasks from the job tracker. Job tracker is periodically contacted by the task tracker to report the completion progress of the tasks and requests for new tasks. Hadoop architecture can be shown on Figure 4.2.

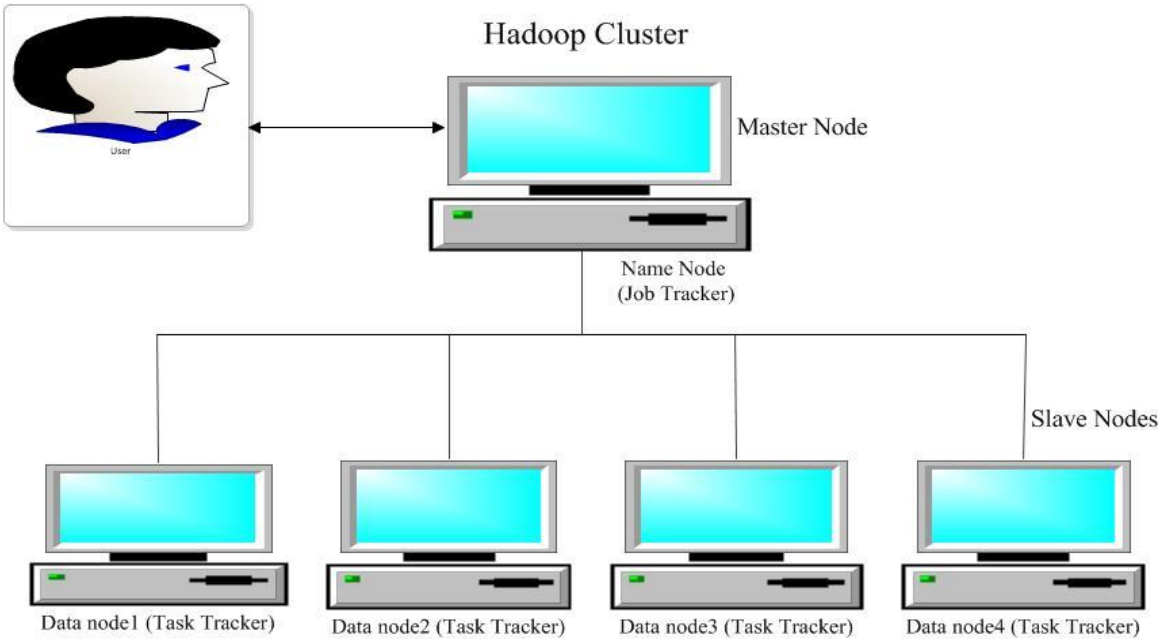

Figure 4. 2: Hadoop Architecture

4.5 R Language

R is an open source programming language and software package widely used by statisticians and data scientists to perform statistical analysis of data [59].R uses several mechanisms for data analysis such as clustering, classification, regression and text analysis. It provides a wide variety of libraries to perform analysis and visualization. These libraries include statistical, machine learning, graphical techniques and time series analysis. Furthermore, R is able to perform additional functions such as data cleaning, data extraction, data loading, data transformation and predictive modelling. Many data scientists use R due to its remarkable features that include effective programming language, data analytics and relational database support. R now can be connected with several data stores like MySQL, SQLite and MongoDB as well as Hadoop for data storage activities. It also can be integrated with Hadoop MapReduce for data analytics [61].

4.5.1 R and Hadoop Integrated Programming Environment

RHIPE is R Hadoop integrated programming environment. It is used to integrate R with Hadoop and MapReduce programming model. It is a software package that allows R users to create Map and reduce jobs completely within R environment using R expressions [61]. It was developed by Saptarshi Guha as a PhD thesis in the Department of Statistics at Purdue University in 2012. To perform data analysis using RHIPE, some software and packages need to be installed as follows:

1. Installing Hadoop.

2. Installing R.

3. Installing protocol buffers.

4. Setting up environment variables.

5. Installing rJava.

6. Installing RHIPE.

RHIPE has some components that used for data analytics as follows:

1. RClient is an R application used to call the JobTracker to execute the job with an indication of several MapReduce job resources such as Mapper, Reducer, input format, output format, input file, output file, and other several parameters that can handle the MapReduce jobs with RClient.

2. JobTracker is the master node in Hadoop MapReduce operations. It is used to initialise and monitor the MapReduce jobs over the Hadoop cluster.

3. TaskTracker is the slave node of the Hadoop cluster. It performs the MapReduce jobs as per the orders given by Job Tracker. It also retrieves the input data chunks and run R-specific Mapper and Reducer over it and then the output will be written on the HDFS directory.

4. HDFS is a distributed filesystem used for data storage over Hadoop cluster. It also provides data services for various data operations. Figure 4.3 shows all components of RHIPE.

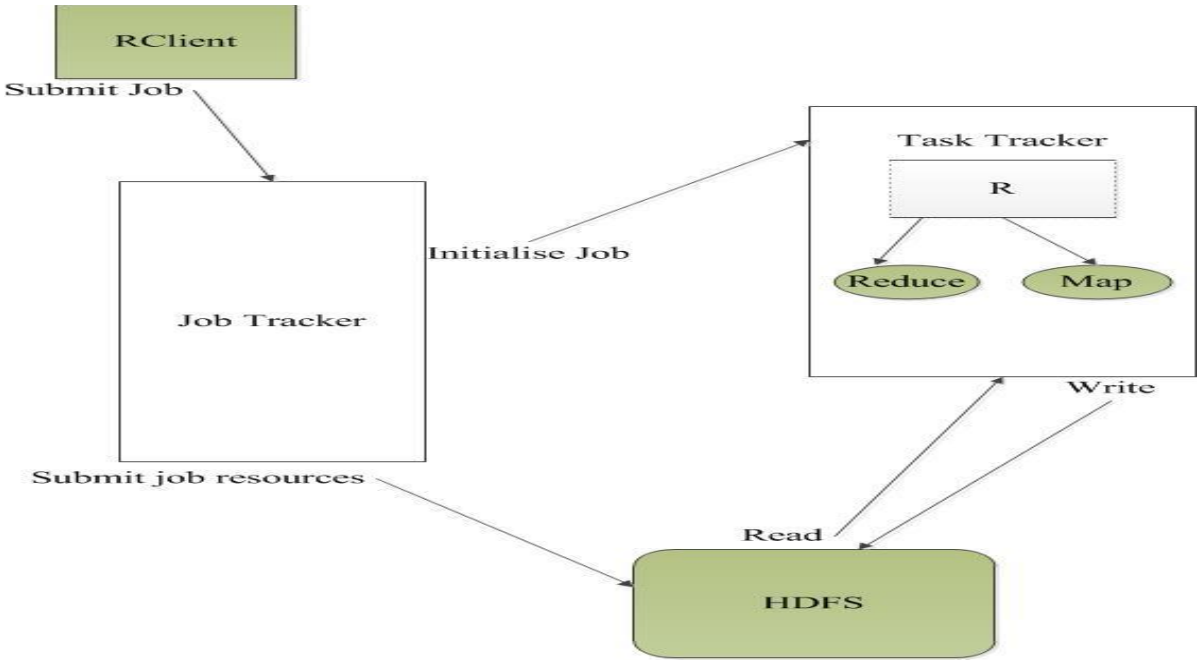

Figure 4. 3: Components of RHIPE

4.6 The Implementation of Solar Radiation Time Series Analysis Using R and Hadoop Integrated Programming Environment

The analysis of solar radiation time series was implemented in the Hadoop MapReduce framework with R language. In this paper, RHIPE is employed to analyse and extract massive volumes of solar radiation data. These datasets are taken from London weather centre for period from (1996-2005) that include monthly mean daily irradiation on inclined planes. We used such old data because it is available for no charge and it is also the most accurate irradiation data since the solar irradiation is estimated. Our proposed work can suit any solar irradiation time series data at any period. The datasets are converted to .csv format (comma-separated values) and then moved directly to HDFS storage for offline analysis. The analysis of solar data includes a calculation of the irradiation of west and south west London using statistical calculations by R language in Hadoop cluster. In this work, R Hadoop integrated programming environment (RHIPE) was employed to perform the statistical calculations on the solar

irradiation. RHIPE package used divide and recombine technique to perform the analysis and calculation on solar data. The solar datasets were automatically divided into several chunks by HDFS. The computation is performed across Hadoop cluster on these chunks by specific R analytics operation. The size of chunks is specified in the cluster configuration file on Hadoop cluster (hdfs-site.xml).

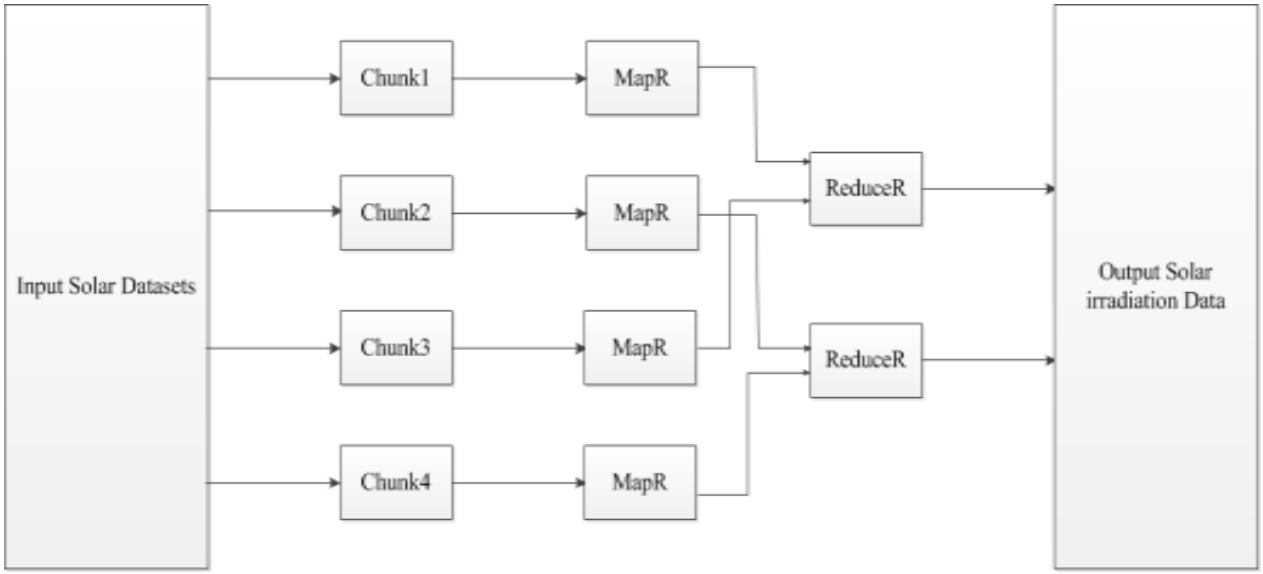

Figure 4. 4: Solar data analytics based on HadoopR

Figure 4.4 shows the whole process of solar data analysis using Hadoop R framework. When HDFS divides the solar datasets into chunks, each chunk of solar dataset is assigned to a Map task. All Map tasks are executed in parallel to process the solar datasets. Number of Map tasks relies on the number of Map slots. Number of Map slots can be specified in the cluster configuration file (mapred-site.xml). In this work, one slot was assigned on each VM. Since the cluster involves 8 VMs, 8 slots were assigned in the cluster and consequently 8 Map tasks were executed in parallel for solar datasets processing. Furthermore, Map slots numbers that are specified to each VM relies on Hardware specifications and processing capacity like CPU cores number and physical memory. Map tasks are responsible to process the data chunks. Each Map task processes the assigned data chunk. When the Map process is finished, all intermediate results will be sent to the Reduce tasks to be processed. Similarly, the number of reduce tasks depends on the number of Reduce slots configured in the cluster configuration

file. Eight Reduce slots were configured and so eight Reduce tasks were executed to obtain the final results. The analysis of solar data is implemented in Hadoop computing environment using R language. Hadoop R framework is fault-tolerant and scalable and provides replication of data chunks and can distribute them on different nodes to avoid any delays and failures. Since Hadoop provides scalability, it can easily add more nodes to increase the computation speedup for the proposed work. It also provides the availability and fault-tolerance. For instance, if any failure happens on the node due to software or hardware issues, the job tracker is responsible to find it and assign the running tasks to another available node.

4.7 Work Evaluation

The evaluation on solar data analysis based on Hadoop R framework (RHIPE) was conducted. A comparison between R language and the Hadoop R framework (RHIPE) have been carried out in terms of computation efficiency and accuracy. The performance evaluation was implemented on irradiation solar data provided by the London Weather Centre (1996-2005). Since these datasets are not very massive, duplication on the datasets was performed to create big data scenario. Datasets were replicated to be tens of Gigabytes.

4.7.1 Cluster Setup

The proposed work was implemented using Hadoop cluster. This cluster involves eight virtual machines and each machine is assigned with 8 GB of RAM and 4 CPU cores with total 300 GB for storage. This cluster was setup on two high performance servers. The analysis on solar data using Hadoop R framework (RHIPE) was performed on eight virtual machines while the analysis based on R language was executed on one machine in R environment. The specifications of Hardware and Software resources are displayed below in Table 4.1.

Table 4. 1: Cluster setup

Intel Xeon X5550 server 1 and uxisvm04 server 2	CPU	32 cores
	Processor	2.27 GHz
	Hard disk	360 GB
	Connectivity	1 GBit Ethernet LAN interconnectivity between two servers
	memory	64 GB
Operating System	Host Operating System	Microsoft windows server 2012 R2
	Guest Operating System	Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)
Software	R, RHIPE 0.65.2, protocol buffer 2.4.1, rjava.	

4.8 Experimental Results

A number of experiments were carried out to evaluate the efficiency of the Hadoop R framework (RHIPE). From Figure 4.5, it can be observed that the RHIPE significantly outperforms R in terms of computation using a single virtual machine. The execution time of solar irradiation data processing is decreased using RHIPE framework, while the execution time is increased using R language. Figure 4.6 shows that the execution time of R goes up when the data size is increased, while the execution time of RHIPE using eight virtual machines keeps constant .The scalability of RHIPE is performed based on Gustafson's law [133]. Furthermore, it was evaluated using various numbers of data sizes and different numbers of VMs. Five different sizes of dataset and varied number of virtual machines from 1 to 8 were employed to evaluate the proposed work. The scalability of the RHIPE work is shown in Figure 4.6. Based on the results shown in Figure 4.6, the calculation of the speedup of the RHIPE can be calculated using

Speedup $(S) = \frac{T_{s\,(S)}}{T_{p\,(S)}}$ (4.1)

Where T_s represents the execution time of solar data analysis based on RHIPE using one VM while T_p represents the execution time of solar data analysis in parallel based on RHIPE using several processing nodes.

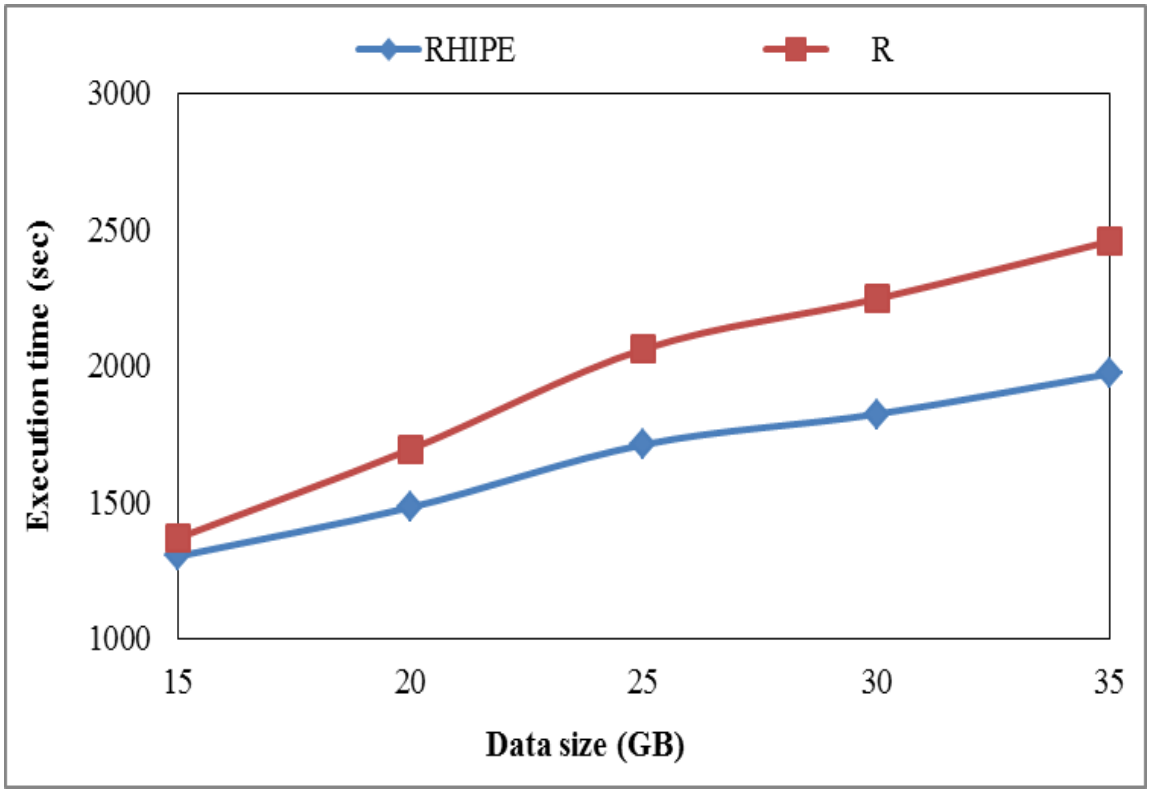

Figure 4. 5: Comparison between RHIPE and R using single virtual node

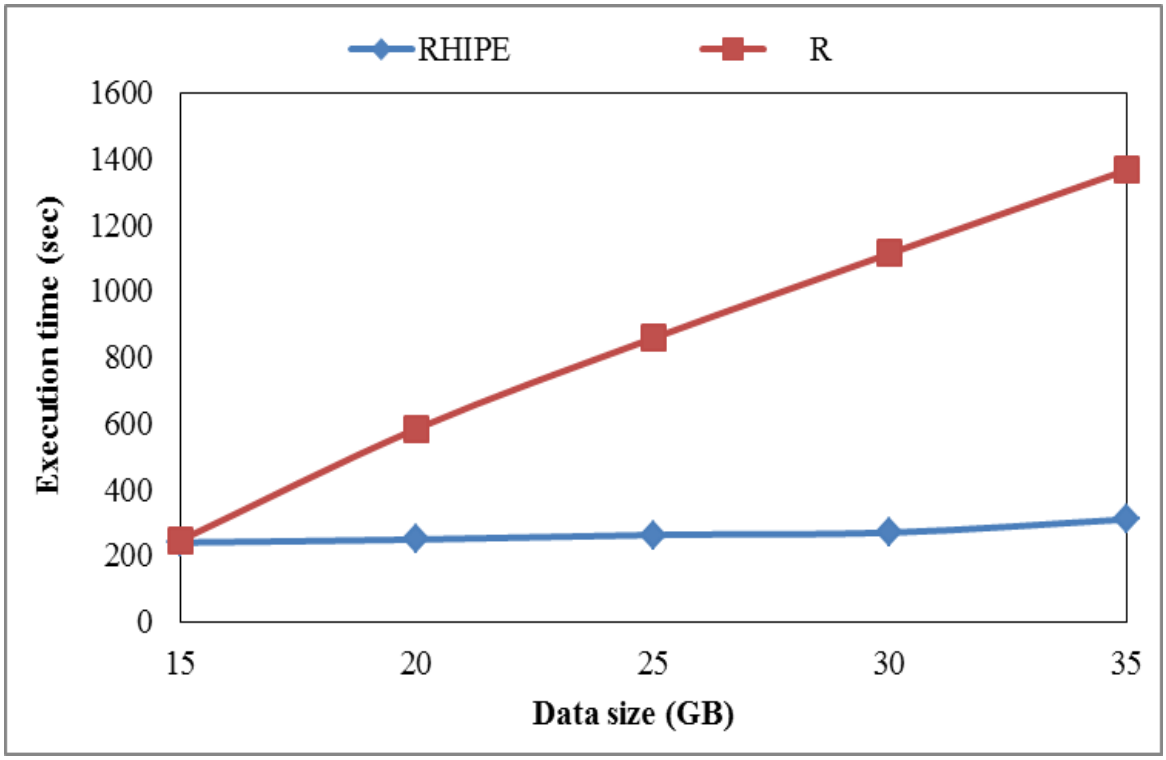

Figure 4. 6: RHIPE Scalability

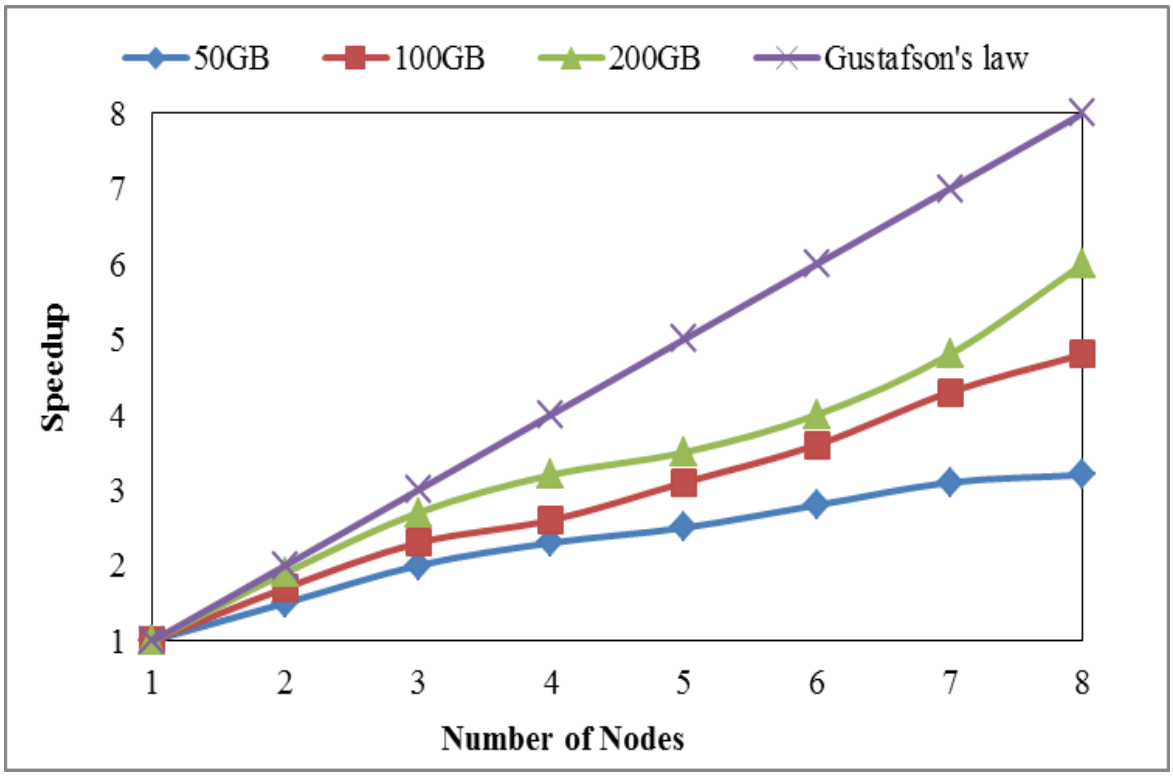

Figure 4. 7: RHIPE Speedup

4.8.1 The Analysis of R and Hadoop Integrated Programming Environment Speedup

The speedup in computation for parallelizing sequential program can be calculated using Gustafson's law as follows:

$$S = f + p(1 - f) \quad (4.2)$$

Where S is the speedup, f is the serial portion of the unscaled workload and p is the number of processing nodes used in the computation process. Gustafson's law is known as fixed-time speedup model. It maintains the execution time is fixed and scales the workload with an increasing number of processing nodes. Figure 4.7 shows the speedup analysis of RHIPE. It can be clearly noted that the speedup goes up with an increasing number of data size. However, as displayed in the Figure, the results never obtain that which is expected by Gustafson's Law. This means that Gustafson's Law is not sufficient to calculate the speedup of a parallelized program which is executed in a cluster computing environment such as Hadoop because the communication overhead of a user job in the cluster computing is not considered by Gustafson's Law. As a result, revision of Gustafson's Law has been done. Equation 4.3 represents the revised Gustafson's Law:

$$S = f + p(1 - f) + R_C \quad (4.3)$$

Where R_C is the ratio of communication overhead to the computation needed for each Hadoop job. High ratio of communication to the computation of parallel program can affect the performance in speedup. Therefore, the ratio of the communication to the computation should be minimized to obtain better performance in speedup in Hadoop cluster. To perform that, the size of data chunks should be large as illustrated in Figure 4.8. In this Figure, four different sizes of data blocks ranging from 8 to 64 MB have been used to evaluate the performance of speedup. It can be clearly observed that the execution time of (RHIPE) goes down when the size of data block goes up. This is because the large size of data chunk produces a small number of tasks that results in a small overhead in communication. Furthermore, the larger data blocks leads to a high workload in computation that leads to increase the speedup according to Gustafson's law.

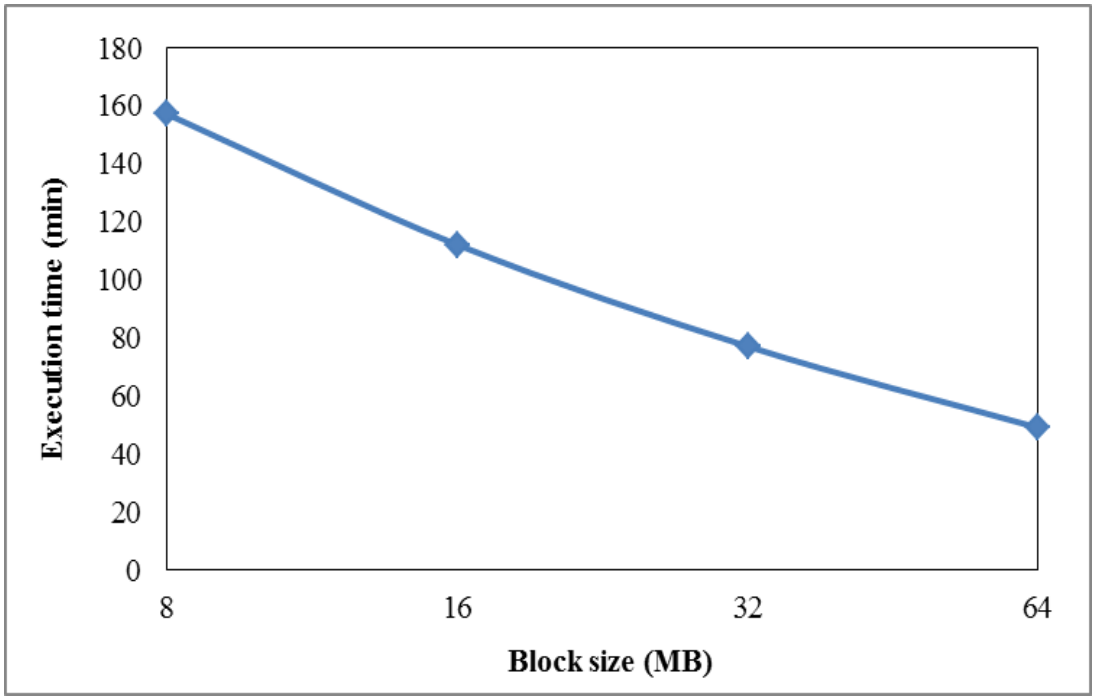

Figure 4. 8: Computational overhead of RHIPE

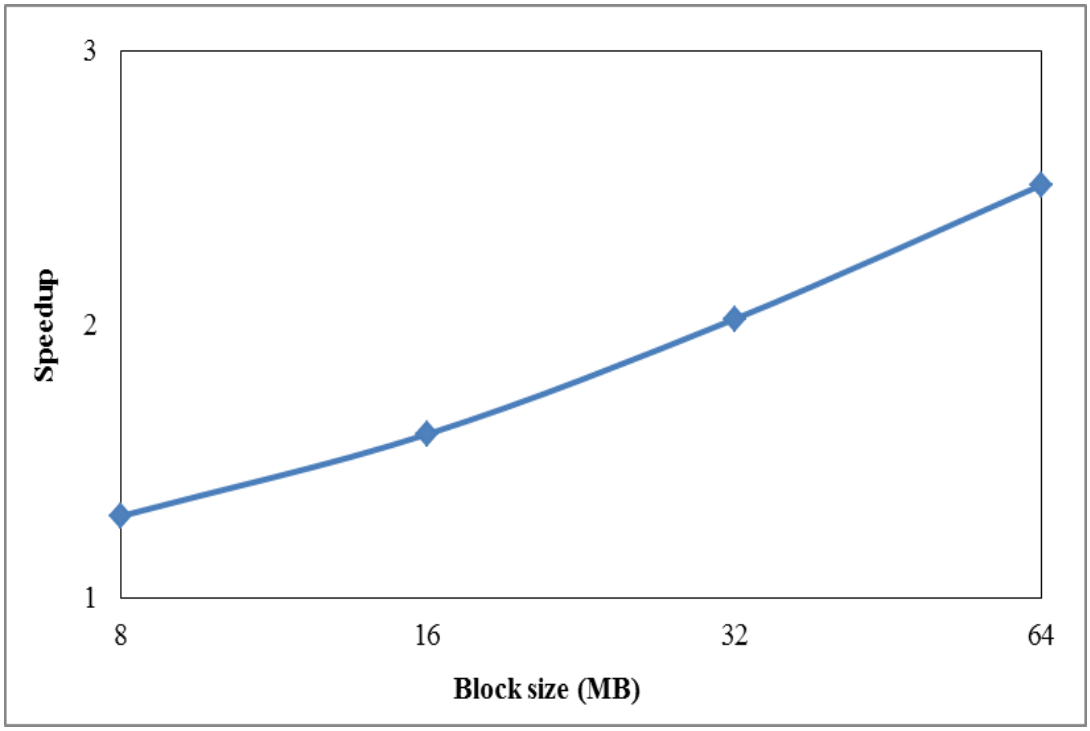

Figure 4. 9: Speedup of RHIPE against data block size

Figure 4.9 shows the speedup of RHIPE using varied sizes of data blocks. From this Figure, it can be seen that the speedup of RHIPE in computation goes up with an increasing size of data blocks. Furthermore, the speedup of RHIPE is 2.2 times faster in computation using 64 than when using 8 MB data blocks, consequently a better improvement in performance can be achieved with larger datasets.

4.9 Summary

This chapter presented R Hadoop integrated programming environment (RHIPE) for the analysis and statistical calculations on solar irradiation. This work was carried out based on an offline analysis of solar data available from the London Weather Centre for the period (1996-2005) that includes monthly mean daily irradiation on inclined planes. The data sets were uploaded into HDFS (Hadoop distributed file system) as a CSV file and were divided into multiple chunks and distributed amongst a cluster of computing nodes. This work was evaluated using eight virtual nodes of Hadoop cluster. The experimental results showed that RHIPE outperformed R in terms of computation, scalability and accuracy. Furthermore, Gustafson's law has been amended to calculate the speedup of the sequential program for RHIPE using parallel computing.

Chapter 5

Routing and Scheduling Algorithm Based On Software Defined Networking for Big Data Applications in Data Centre Network

We live in the era of internet of things, big data, virtualization and cloud computing technologies. However, the current network infrastructures of these technologies lack scalability, cost effective maintenance, centralized management and fault tolerance. As a result, the emergence of software defined network (SDN) provides many network facilities from the aspect of intelligent and centralized management, scalability, less operation cost and less complex maintenance. For intensive big data applications like MapReduce in a data centre network, SDN provides an intelligent and dynamic control for data traffic to improve the performance of big data jobs. Heavy traffic is generated during the shuffle phase of MapReduce jobs, when the output intermediate data of mapper nodes is transferred to the reducer nodes leading to increase the execution time of the shuffle phase. In this chapter, SDN is employed to reduce the shuffling time by proposing an effective scheduling and routing algorithm to control each shuffling flow in the data centre network. The proposed algorithm is to provide efficient network bandwidth for shuffling flows according to their network demand as well as their size and number. It also prevents the congestion by assigning efficient alternative routes based on the network bandwidth utilization of each path in the data centre network as well as the size of each routed shuffling flow.

5. 1 Introduction

Nowadays, vast amount of data is generating everywhere due to the rapid development of cloud services and the wide use of internet of things (IoT) applications. Moreover, large datasets are also generated from different sources such as sensors, social networking, video streaming and online services. This leads to an explosion of data, which is considered as a big data that cannot be captured, analysed and processed using the traditional tools. These massive datasets require enormous parallel processing and distributed computation. As a result, powerful platform such as Hadoop MapReduce can be implemented based on a cloud computing environment to utilize massive parallel

processing and distributed computation for the large datasets. Hadoop [21] is scalable computing platform that enables parallel processing and distributed storage across number of servers and computing nodes. It is also fault tolerant and able to execute the code of MapReduce directly close to the data that operates on to limit the data transfer within the cluster. However, large data blocks are transferred between mappers and reducers during the shuffling phase, which consumes considerable traffic that requires appropriate network resources. Moreover, computing resources such as memory, CPU, storage and disk I/O requirements can be scaled with number of servers; however, scaling the system will lead to increase the network traffic in the underlying network. Furthermore, massive collections of datasets are exchanged between servers in a data Centre network. As a consequence, many flows among servers are generated to transfer these massive datasets during the processing phase. Transfer such large datasets quickly and process them efficiently needs a convenient bandwidth allocation for each flow. It may be argued that the networking part for a Hadoop cluster still bottleneck even with the optimal configurations of the computing part. A previous study on Facebook showed that the communication phase consumes 33% of the running time of jobs and some cases even reaches to more than 50% of the running time [134]. SDN can be used to improve the performance of the networking part of Hadoop MapReduce jobs, especially during the shuffling phase. It has some remarkable advantages such as direct network programming, centralised management, intelligent control, agility, flexibility, network abstraction as well as CapEx and OpEX reduction. In this work, a dynamic and effective approach based on SDN has been proposed to improve the performance of Hadoop MapReduce jobs by reducing their shuffle phase time.

5.2 Related Work

Hadoop is the most popular framework of distributed storage and parallel processing for massive datasets, which generate from different sources. Many research works have been carried out to improve the performance of Hadoop MapReduce. In [135] Narayan proposed the integration of SDN technology and Hadoop. The main idea of the proposed work is to identify the traffic of Hadoop intermediate data and the background traffic by using the flow rules and then apply different quality of service (QoS) for them. The experimental results of this work showed that the execution time of

MapReduce job went down due to the sufficient amount of bandwidth allocated for the shuffle traffic. However, this work is only suitable for small scale cluster and cannot be applied to large scale clusters in data Centre network with large number of switches and servers. The work proposed in [68] presented an application-aware SDN routing scheme for Hadoop to speed up the data shuffling of MapReduce over the network. Another work was proposed in [87] to improve the job completion time. This work proposed application-aware network in SDN (AAN-SDN) for Hadoop MapReduce to provide both underlying networks functions and MapReduce particular forwarding logics. Flexible Network framework (FlowComb) was proposed in [136] for big data applications to achieve high bandwidth utilization and fast processing time by predicting the network application transfers. Yi Lin and Yu Liao [137] used an SDN app for Hadoop cluster to speed up the execution time of MapReduce jobs. The proposed work implemented the SDN app in Hadoop cluster for easy deployment of flow rules for Hadoop applications. However, the work has evaluated small cluster with one physical switch and have not investigated the performance of Hadoop jobs in large clusters in data Centre network.

The work presented in [85] focuses on enhancing the data locality of Hadoop in a global way and assigning tasks efficiently by proposing bandwidth-aware scheduling with SDN in Hadoop cluster. It exploits SDN capabilities for jobs scheduling of big data processing. Jin et al. [86] Proposed different approach to reduce the running time of a Hadoop job. The proposed approach is based on an initial task allocation produced by balance reducer (BAR), and then the completion time of a Hadoop job can be progressively decreased by the initial task allocation. SDN is employed to build big data platform for social TV analytics in [138]. The proposed system integrated SDN with Hadoop to enable intermediate data transfer among different processing units to achieve fast data processing rate. SHadoop is presented in [33] to improve the performance of Hadoop MapReduce jobs by optimizing the setup and clean-up tasks of a Hadoop job. It also provides massaging communication mechanism to obtain rapid performance of task scheduling and execution. In [139] Hedera is proposed based on SDN to replace congested flow path for data transfer with less congested flow paths by using centralized controller. However, setting new flow path instead of the existing path for flows leads to the loss and reordering of packets. Some other work like [140] proposed multiple paths to split and transmit large flows to obtain high throughput by splitting the

flows at switches. However, this process is a challenging task because the switch lacks the ability to match the sequential number of TCP.

5.3 Software Defined Networking

SDN is an emerging network architecture approach that provides dynamic, manageable and cost-effective platform to enable networks to be intelligently and centrally programmed. It is adaptable platform for many network applications. It decouples the network's control logic (control plane) from the data plane. The separation of the control and data planes brings direct programming and central management of networks using the SDN controller. SDN allows network administrators to provide abstraction of lower level functionality and enable the programming of network behaviour dynamically via well-defined open application programming interface(APIs), which includes both of northbound and southbound APIs that represent the communication channels between SDN layers [141]. We used Openflow protocol to enable the SDN controller to interact with the forwarding plane and manage the configuration state of switches and routers [142] . Figure 5.1 shows the architecture of software defined network. The following layers define and explain the architecture of SDN [103].

1- Application layer is the layer that includes SDN applications, which are programs that directly communicate and exchange information with the SDN controller via northbound APIs. Furthermore, applications can construct an abstracted view of network infrastructure by collecting information from the SDN controller for their decision making purposes.

2- Control layer (SDN controller) is the core of the SDN network that manages the flow control between the SDN networking devices via southbound APIs and the SDN applications via northbound APIs to provide intelligent networking.

3- Network infrastructure layer is the layer that includes the SDN networking devices that control the forwarding process of data path. All forwarding and routing decisions will be executed based on the flow tables provided by the SDN controller via the southbound APIs.

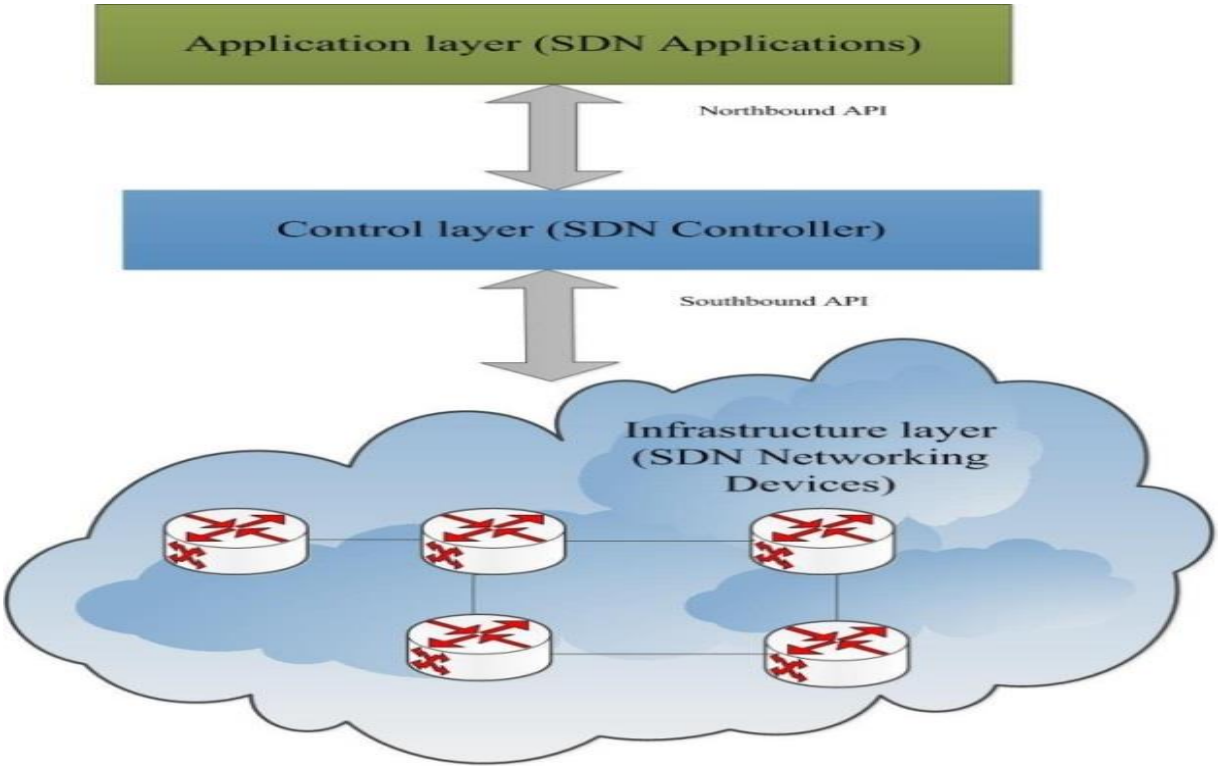

Figure 5. 1: SDN architecture

5.4 Routing Techniques and Network Topologies

Before the discussion of our proposed work, it is important to explain some routing techniques, like ECMP and some data centre network topologies. Multipath techniques are widely used in the modern data centre network for forwarding and distributing flows across multiple paths so as to achieve better bandwidth utilisation. ECMP [71] is used to distribute the flows across multiple equal cost paths to exploit the full capacity of network bandwidth. However, it has some limitations, such as the static scheduling of flows across multiple paths. That is, it uses a hashing value policy to allocate flows with certain paths. It also lacks a global view of the entire network, missing its current load as well as the individual characteristics of flows and their future network demand. As a result, we propose in this paper an effective routing algorithm based on application level information to estimate the demand of all shuffling flows during the MapReduce process, as explained in section 5.5. The characteristics of the most popular topologies of three-tier architectures, like fat tree topology, have been studied. From this study, we have identified some limitations and bottlenecks of this topology. Fat tree topology [143] is divided into multiple pods, with each including the switches of the edge and

aggregation layers. The connection inside the pod is considered as a local pod connection, because the traffic remains inside it. On the other hand, the connection between different pods is considered as a remote connection, because the traffic of connecting pod passes through one or more core switches. This hierarchical architecture limits the locations of end hosts and also creates loops in the network due to the redundant paths that connect the end hosts when multipath techniques are used, like ECMP. As a result, a spanning tree [144] is used to prevent loops by selecting a single path and disabling all other redundant paths. However, this routing scheme of a spanning tree leads to poor network utilisation, because the flows in the data centre network will employ few paths and leave others redundant, only reutilising them in the case of any outage or failure. We illustrate some examples of flow transfer based on fat tree topology.

There are three cases of transfer flows between two hosts in a data centre based on fat tree topology. The first, involves sending shuffling flow from host 1 to host 2. In this case, there is only a single path between them, because both hosts are located in the same rack. Hence, all possible paths between the two hosts go through edge switches only and the generated traffic remains inside the rack, with there being no need to traverse any aggregation switches. In the second case, host 1 sends its flow to host 4, which is located in a different rack, but within the same pod. In this case, the connection between them is an intra-pod connection, because all the possible paths between these two hosts will pass through edge and aggregation switches. The third case is in relation to transferring flows between two hosts located in different pods, such as host 2 and host 8. In this case, there are multiple paths between them to transfer flows. However, the produced traffic between the two hosts has to traverse edge, aggregation and core switches, because each host is located in a different pod and all possible paths should go through different core switches. The situation becomes more sophisticated when some hosts in different pods exchange shuffling flows at the same time and might contend for the same links, especially in the aggregation and core switches, thus creating congestion that makes the bandwidth utilisation of the core and aggregation links becoming over-utilised. It is supposed that multiple hosts exchange their flows at the same time. Specifically, host 2 sends its flow to host 6, host 10 sends its flow to host 16 and host 5 sends its flow to hosts 15, respectively, all simultaneously. Figure 5.2 illustrates the path between hosts 2 and 6 as well as that between hosts 10

and 16 in bold lines. It is observed that there are multiple paths between all the hosts. However, it is noted that there is a challenge to assign even a single path among the multiple paths in the data centre network for hosts 5 and 15 because of the congestion that has occurred in the network. The main cause of this is the architecture of fat tree topology that constrains the location of end hosts. Since host 5 is located in pod 2 and host 15 is in pod 4, it is a challenging task to assign a path between the two hosts even though we selected the right side of pod 2 to avoid the overlapping. It is impossible to avoid the overlapping in pod 4, because the right side in pod 2 can only reach the left side of pod 4 and consequently, this creates congestion between the two hosts. As a result, it has become crucial to design an efficient type of data centre architecture, like leaf-spine topology [145]. Unlike fat tree topology, this consists of two layers. The first is the leaf layer that includes several switches connected to end hosts in the network. It is connected to the spine layer that represents the second or top layer. Leaf-spine topology is widely adopted in large data centres and cloud networks due to its remarkable features, such as scalability, reliability and effective performance. However, applying multipath algorithms, such as ECMP, as a forwarding technique for shuffling flows to utilise more bandwidth in the leaf-spine topology is not an effective way, because it is a static scheduling algorithm and it does not consider the network utilisation or flow size.

For instance, there are three different hosts in the same rack, which are connected to the same switch in the leaf layer transferring their flows to other hosts in different racks. The first case, is when host 2 sends its shuffling flow to host 8, whilst the second, is when host 4 sends its shuffling flow to host 6 and the third case is when host 3 transfers his shuffling flows to host 10, as shown in Figure 5.3. We observed that host 3 might compete for the same heavy loaded link in the leaf switch, because of the allocation technique of ECMP, whereby it might choose the same heavy loaded link for two large shuffling flows, thus resulting in a congestion and collision. The reason for this, is because, as aforementioned, ECMP lacks a global view of entire network. Moreover, with ECMP algorithm, the flow is routed based on its hash value. Hence, flows might result in using the same path and creating congestion in some links in the leaf and spine switches. It is also seen from Figure 5.3, that all possible paths of shuffling flows for all cases might compete for the same leaf and spine switches, which leads to overload on some link switches. Furthermore, crashing or failure might occur on some links that

belong to the allocated path for shuffling flows in the leaf and spine switches. As a consequence, we propose an effective routing algorithm based on SDN that performs the routing process, which respects the network resources demand of each shuffling flow as well as their size and number. The proposed algorithm is also able to reroute the shuffling flows to another available path in the case of any failure or crashing on any link in the network. The proposed algorithm is explained in the following section.

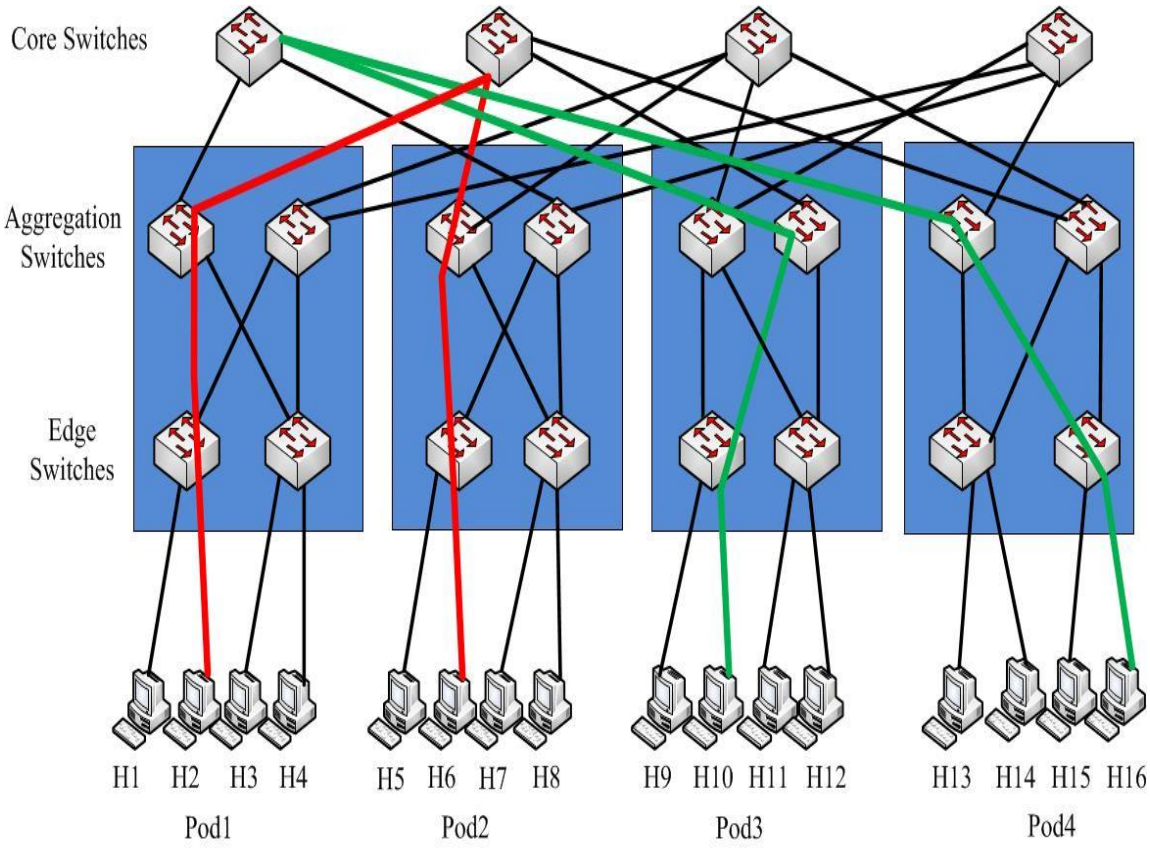

Figure 5. 2: Path allocation challenging in fat tree topology

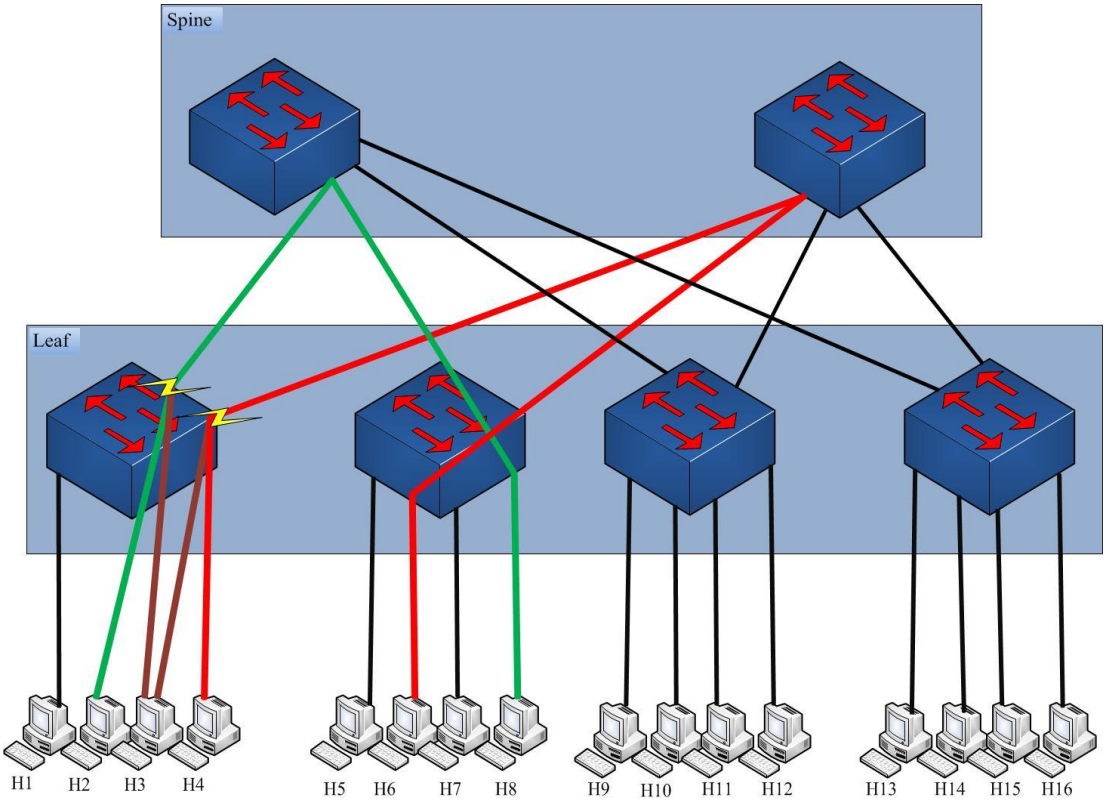

Figure 5. 3: Path allocation using ECMP in leaf-spine topology

5.5 The Implementation of the Proposed Work

Our proposed work consists of three modules as follows.

1- Link monitor module: This module monitors network link status, such as link loading in the network and computes the link weight. It periodically gets the statistics information of all links loaded in the data centre network from all the connected OpenFlow switches at specific intervals. Statistics such as per-table, per-flow and per-port are collected and stored as snapshots. All switches in the network are connected to the SDN control. However, the SDN controller lacks the required information of all links between the switches and hence, a link layer discovery protocol (LLDP) [146] is used to identify the needed information of all links and the switches layer in the network topology. Statistic information about links loading is used by the routing module to calculate the paths accordingly. The current load of each link in the data centre network is computed by using N transmitted bytes from the port within recent interval t over the bandwidth (B) of the link. The formula below calculates the current load of the link:

$$L_{Lk} = \frac{total\ number\ of\ transmitted\ bytes\ within\ t}{B} \quad (5.1)$$

It is supposed that all links have the same bandwidth and each link has fixed weight. In our case, we propose that the link weight (W) is 1. It is very important to check whether the current load of each link (L_{Lk}) reaches or does not reach the peak of link depending on the link weight (W) by comparing it with (L_{Lk}). If L_{Lk}<1, it means that is has not yet reached the peak of the link. However, if L_k=1, it means that it has and this may cause link overloading, because of some heavier flows and consequently, result in improper path allocation. Hence, the weight of each link should be estimated based on the number of flows and the throughput of each. The natural demand of shuffling flows is estimated by the Hadoop engine module. It is worth noting that the current load reaches the link capacity, if it exceeds threshold γ which has been set to be 90% of the link capacity. Furthermore, we compute the path load for all flow paths in the leaf and spine switches by using the maximum load of each link, which belongs to the path as explained in Equation 5.2.

.

$$L_p = \max_{l \in p} l_l \quad (5.2)$$

Where, (p) is defined as the path used to route the shuffling flow from source to destination. Each link that belongs to the path (p) is represented by (l) and (l_l) pertains to the load of each link that traverses the path at the leaf and spine switches from the source node to the destination node. Once the path load of each link is computed, all information is delivered to the scheduling and routing component to select the convenient path that has the least path load (L_p). Then, it installs the flow entries into a set of switches of the selected path.

2- Hadoop monitor engine: In a Hadoop cluster environment when the map task in the mapper node writes its output data to the reducer node, shuffling traffic is generated during the shuffle phase of a Hadoop job. This traffic needs sufficient network bandwidth to accelerate the processing time of the Hadoop job. However, the main Hadoop framework does not contain sufficient information about the required network resources. Therefore, this module is proposed to identify the data transferred from the mapper node to the reducer node in a Hadoop cluster during the shuffle phase of a

Hadoop job. The data is transferred through a number of flows during the shuffling phase.

This module is responsible for recording all the required information of these flows from all the connected Hadoop servers. In a Hadoop cluster, as aforementioned, there is one job tracker and several task trackers. The job tracker is responsible for monitoring the progress of Hadoop jobs by receiving heartbeat messages from each task tracker, but these messages do not include information about the network resources. To obtain such information, a software engine has been installed on each Hadoop server. This engine detects when a map task has finished and starts to send its shuffling information to the reducers, whilst then recording the size of the map output data, which is transferred over the flow to the other reducers. After this process, the Hadoop engine will obtain the required network bandwidth for each shuffling flow. It maintains a table that contains all shuffling flows with their networking demands. Furthermore, all the collected shuffling information includes the source IP address, destination IP address and the size of each shuffling flow. The Hadoop monitor engine also determines the total amount of shuffled data and the number of shuffling flows transferred over each link. All information about shuffling flows is delivered by the Hadoop monitor to the scheduling and routing module to assign proper paths, according to the bandwidth needed for each shuffling flow and the current load of link utilisation.

3- Scheduling and routing module: In the forwarding module of the OpenFlow floodlight controller, a packet-in message is generated to notify the controller that new flows have arrived at an OpenFlow switch. The switch checks the packet and if there is no match with its flow entries, the packet is forwarded to the controller. On the other hand, a flow-removed message is also generated when a flow expires in an open flow switch. In this work, a scheduling and routing module is proposed to assign efficient paths for the exchangeable shuffling flows between different hosts in the data centre network.

This module performs the scheduling and routing of the shuffling flows on the chosen paths and it has two tasks. The first is the calculation of the possible paths based on the statistics from the link monitor module that includes the loads on all links in the network. It also uses the collected information by the Hadoop monitor engine to compute the possible paths of different shuffling flows. The collected information by

the Hadoop engine module contains a list of shuffling flows including source/destination IPs, flow size and transfer volume over each link. All this information is recorded in a network table to be used for the calculation of the path load in the routing process. This table also contains scheduled flows and available capacity for each of them. Once all the information of shuffling flows has been received by the scheduling and routing module, it will compute the possible paths with low load based on the information collected from the link monitor module and Hadoop monitor engine. The second task is to assign efficiently the best possible paths for all shuffling flows, according to the bandwidth needed for each flow.

A scheduling and routing algorithm based on SDN is proposed to obtain an effective routing technique for shuffling flow, according to network utilisation and flow size by computing the current load of all possible paths in the leaf and spine switches. Once the current load is determined according to Equation 5.2, the shuffling flows are routed onto the proper paths. Our proposed work moves the large shuffling flows from heavy loaded links to lightly loaded ones so as to prevent congestion. What is proposed is demonstrated in Algorithm 1 and 2.

Algorithm 1

1- **For** each shuffling flow (*SF*) **do**

2- Collect *SF* size and its network resources demand from the SDN controller

3- Compute the current load of all possible paths for each *SF* according to Equation 5.1

4- Compare the size of each *SF* with the current load of all possible paths

5- Choose the shortest available path for *SF* and check

6- **If** the link of shortest path is active and its current load does not override the pre-defined threshold **then**

7- Keep *SF* routing on this path;

8- **Else**

9- **If** there is any failure in the link of the shortest path or its load exceeds the pre-defined threshold **then**

10- Choose another available path with light loaded or unused links calculated by Equation 5.2;

11- Re-route the shuffling flow on new chosen path;

12- **End if**

13- **End if**

14- **End for**

In this algorithm, the size of the shuffling flow and the demand of the network resources are determined using the Hadoop engine module. This module sends all the required information to the SDN controller. After that, the current load of all possible paths of each shuffling flow between any two hosts is computed using the information received from both the link monitor and Hadoop engine model, as mentioned before. Then, the shortest path with minimum load will be chosen. If the link of the shortest path is active and there is no failure or congestion, the routing of the shuffling flow is kept on this path. However, if there is any crash in the link or its current load exceeds the pre-specified threshold, which is set to 90% of the link capacity of this path, as mentioned for the link monitor module, then another unused or light loaded shortest path should be chosen. This is also computed based on Equation 5.2 and the information received from the Hadoop engine and the shuffling flow is rerouted accordingly. It is worth noting that the SDN controller receives all the required information of link loading for all the Open vSwitches in different layers from the link monitor module, as detailed above.

Algorithm 2

1- **For** each shuffling flow (*SF*) **do**

2- Collect *SF* size and its network resources demand from the SDN controller

3- Compute the current load of all possible paths for each *SF* according to Equation 5.1

4- Compare the size of each *SF* with the current load of all possible paths

5- Choose the shortest available path for *SF* and check

6- **If** the shuffling flow (*SF*) size does not override the pre-defined threshold of link switches of this shortest path **then**

7- Keep *SF* routing on this path;

8- **Else**

9- **If** the load of *SF* exceeds the pre-defined threshold **then**

10- Choose another available path with light loaded or unused links calculated by Equation 5.2;

11- Re-route the shuffling flow on new chosen path;

12- **End if**

13- **End if**

14- **End for**

In algorithm 2, the size of the shuffling flow and the demand of the network resources are determined using the Hadoop engine module. This module sends all the required information to the SDN controller. After that, the current load of all possible paths of each shuffling flow between any two hosts is computed using the information received from both the link monitor and Hadoop engine model, as mentioned before. Then, the shortest path with minimum load will be chosen. If the shuffling flow does not exceed the current load of this shortest path, its routing is kept on this path. However, if the shuffling flow exceeds the pre-specified threshold, which is set to 90% of the link capacity of this path as mentioned for the link monitor module, then another unused or light loaded shortest path should be chosen. This is also computed based on Equation 5.2 and the information received from the Hadoop engine and the shuffling flow is rerouted accordingly. It is worth noting that the SDN controller receives all the required information of link loading for all the Open vSwitches in different layers from the link monitor module, as detailed above.

5.5.1 Performance Evaluation

The proposed work is evaluated using EstiNet emulator because of its scalability and the correctness of performance results [21]. The software emulated a leaf spine topology consisting of 12 switches using 1Gbps links. It includes two layers, which are spine and leaf. The leaf layer is responsible to provide the connectivity to endpoints

such as computing nodes, storage devices, firewalls and other endpoints devices. On the other hand, the spine layer provides the connections between leaf switches. Each switch in the leaf layer connects to every switch in the spine layer. However, the switches in the spine layer are not interconnected with each other. Similarly, the switches in the leaf layers are also not interconnected with each other. EstiNet software also emulates fat tree topology, which includes three layer switches using 1Gbps links. The three layers contain 20 4-ports switches. Both of edge and aggregation layers are configured with 16 switches divided into 4 pods and 4 switches is configured for core layer. Apache Hadoop is installed on 16 hosts using two Linux servers managed by Microsoft azure. All hosts are connecting to the emulated leaf-spine topology by Estinet software .The traffic generated from each Hadoop host goes inside the emulated network. Word count and Tera Sort applications have been run on top of Hadoop hosts. All switches forward packets according to the flow rules received from the floodlight controller. These switches are connected to the Floodlight controller using TCP connection. Another TCP connection is established between the floodlight controller and the Hadoop engines of Hadoop hosts to collect all required information of shuffling flows. The proposed algorithm of our routing scheme is compared with ECMP and TRILL (Transparent Interconnection of Lots of Links) [22]. The performance of the proposed work is measured by using the following metrics: (1) the execution time of shuffling phase using different sizes of datasets (2) the execution time of shuffling phase using different numbers of reducers. The execution time of shuffling phase in leaf spine topology has also been compared with fat tree topology under different number of reducers. The experimental results and discussions are explained in sections 5.5.2 and 5.5.3, respectively.

5.5.2 The First Experimental Results and Discussion

Two experiments were carried out using Word count and Tera Sort applications to evaluate our proposed work. In the first experiment, the performance of our proposed algorithm is evaluated using fat tree topology and leaf-spine topology. The second experiment evaluates the performance work against ECMP and TRILL in the leaf-spine topology. Furthermore, the two applications have been run under different scenario. In the first scenario, different input data sizes ranging from 1-5 GB have been used. The

second scenario employs various numbers of mappers and reducers. Figure 5.4 shows the execution time of a Hadoop job for the Tera Sort application using both the fat tree and leaf-spine topology based on ECMP under different number of reducers. From this Figure, it can be clearly observed that the execution time of shuffling flows in the leaf-spine topology can be reduced, when compared to the fat tree topology. The reason for this is that fat tree topology is mainly designed to process north-south traffic (i.e form the core switches to the edge switches).

On the other hand, the traffic between hosts (west-east traffic) in the fat tree topology is representing a challenging task, because some hosts in the network might connect to the same port and then compete for bandwidth, which results in a delay in the response time. Furthermore, the communication between two hosts in the fat tree topology needs to traverse through a hierarchical path from the edge layer to the core layer, thus resulting in latency and traffic bottlenecks. Figure 5.5 shows the execution time of Tera Sort application for different data sizes using different algorithms based on the leaf-spine topology. Our proposed algorithm was compared with different techniques such as ECMP and TRILL. It is clearly seen that the execution time of shuffling flows are reduced for different data sizes using our proposed algorithm comparing to ECMP and TRILL. The reason is because of the inconvenient path allocation for shuffling flows by ECMP and TRILL. TRILL is mainly designed to solve the issues of Layer 2 spanning tree that utilizes only single path during the communication between any pair of hosts, whilst other redundant paths are only utilized in the case of any crashing or failures. However, TRILL is not able to support multipath routing at layer 3, where only a single router will be active with virtual router redundancy protocol (VRRP). Moreover, it has some limitations like VLAN scale, where the spanning of VLANS is eliminated in the network. VLAN segments in each leaf switch are not accessible by other leaf switches in the network and consequently, constraints the mobility of virtual machine within the data centre. Spanning tree topology might be appropriate for conventional business networks, where few paths are used to exchange the traffic between hosts. However, it is not appropriate for the modern data Centre networks, where high traffic is generated, because it is unable to utilize the full capacity of the network bandwidth and thus leads to the throughput degradation.

Therefore, multipath forwarding techniques such as ECMP is employed to utilize the overall capacity of the network bandwidth in the modern data Centre network. ECMP

can forward the shuffling flows to multi paths equally based on hashing policy. Each packet belongs to a single flow is assigned with the same hashing value in the packet header. However, ECMP lacks a global view of the entire network. The bandwidth needed in the network, future traffic demand and the size of each shuffling flow is not considered in ECMP as it uses static routing process for flow scheduling. Moreover, ECMP follows a distributed scheme at each host that leads to utilize some overloaded links, while leave other light loaded links are unutilized. Our proposed algorithm schedule and route the shuffling flows dynamically, according to their bandwidth required as well as their size and number.

Our observation showed that the shuffling execution time goes up, when number of reducers increase in ECMP and TRILL due to the high traffic generated, as shown in Figure 5.6. On the other hand, the shuffling execution time in our proposed algorithm remains constant, because of the effective utilization of the network bandwidth and efficient dynamic path allocation for shuffling flows. However, the execution time went up, when using 16 reducers due to the over-utilized of CPU, which leads to the resources overhead and hence, increases the execution time.

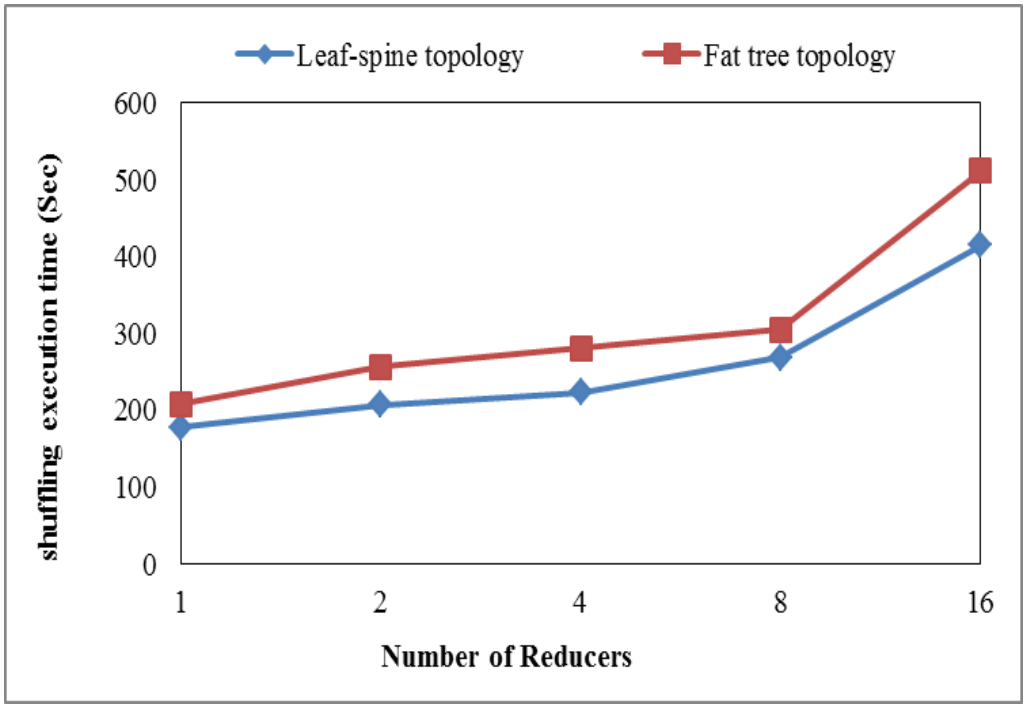

Figure 5. 4: shuffling execution time of Tera Sort using different number of reducers for leaf-spine and fat tree topology

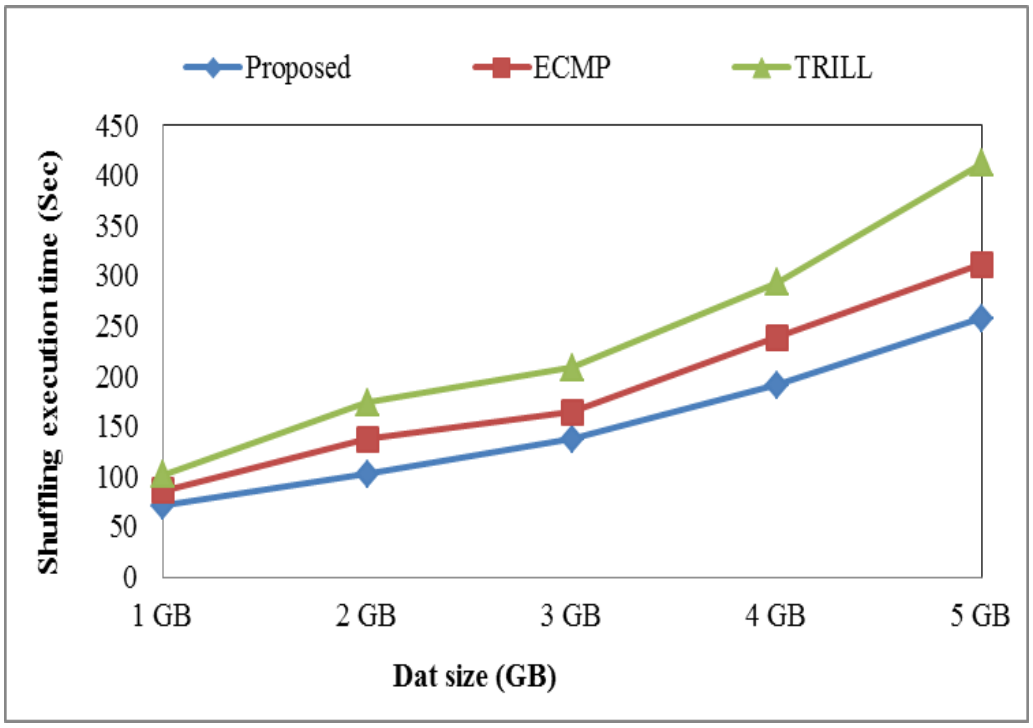

Figure 5. 5: shuffling execution time of Tera Sort

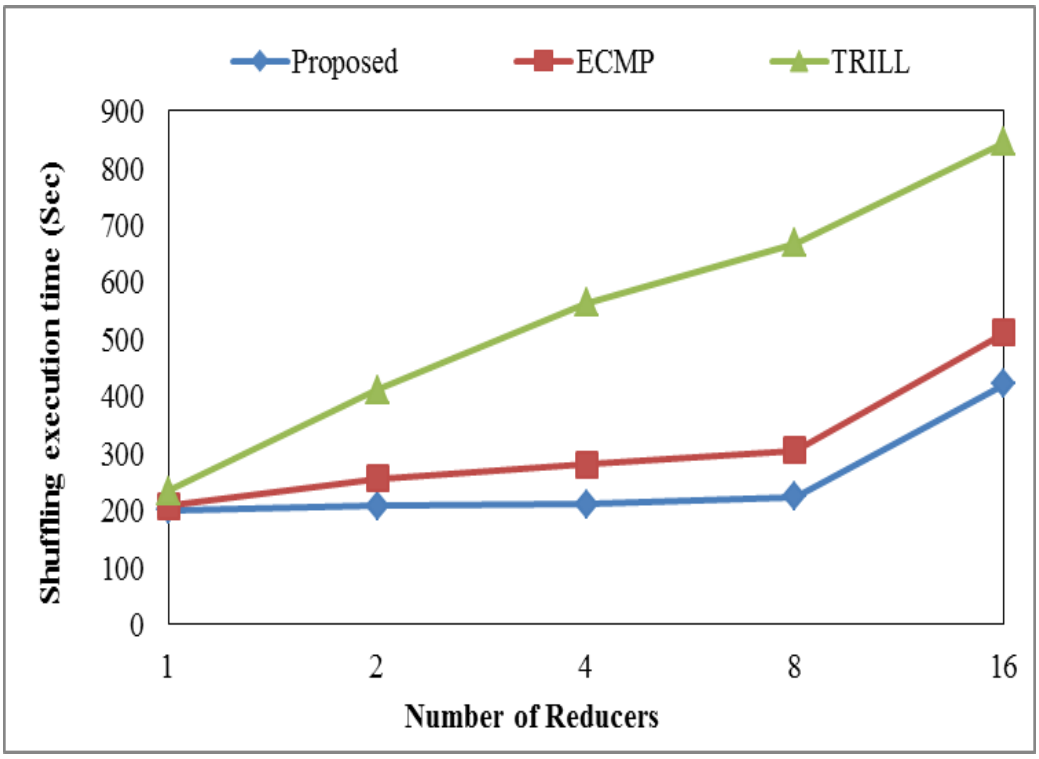

Figure 5. 6: shuffling execution time of Tera Sort using different number of reducers

5.5.3 The Second Experimental Results and Discussion

Two experiments were carried out on a Hadoop cluster based on SDN in the data centre network to evaluate the performance of the proposed work. The performance of the proposed work in these experiments is evaluated based on the following metrics (1) Routing convergence time using different topology sizes (2) Hadoop execution time of Word count and Tera Sort applications within leaf-spine topology under different topology and data sizes. In the first experiment, EstiNet emulator software was used to build two different topologies, namely, fat tree and leaf-spine. In both SDN and conventional networks, three layers of switches were used for fat tree topology. The first was the edge layer, which was assigned with eight switches at the top of the rack. The middle layer or aggregation layer was also allocated eight switches and finally, four switches were used for the core layer. The emulated leaf-spine topology consisted of two layers, with the bandwidth of all the links in the SDN and conventional network being set at 10 Mbps, whilst the link delay was 1ms. 16 Hadoop nodes were used, with each being allocated four CPU cores and 8GB of RAM.

All the Hadoop hosts were connected to the emulated fat tree and leaf-spine topology using EstiNet emulator software. The traffic produced by each Hadoop host went into the emulated network. The previously utilised two real application programs, namely Word Count and Tera Sort were used to evaluate the work performance. All switches in the emulated fat tree and leaf-spine topology were connected to the SDN (Floodlight) controller using a TCP connection. Another TCP connection was deployed to connect the floodlight controller to the Hadoop engine. The fat tree topology of data centre network based on SDN was compared with the conventional network to evaluate the performance of Hadoop MapReduce jobs. The leaf-spine topology based on SDN was also compared with the conventional network. Open shortest path first (OSPF) was used for the conventional network. Figure 5.7 shows the fat tree topology of the proposed work based on SDN using 20 switches. The leaf-spine topology with 12 switches is shown in Figure 5.8.The second experiment also involved the same software emulator in the first experiment, but with different network topology size, as shown in Figure 5.9. In both the SDN and conventional networks, eight switches in the edge and aggregation layers and only two in the core layer were used. On the other hand, six switches were

used in the leaf-spine topology for the SDN and conventional network, as shown in Figure 5.10. The evaluation of the proposed method using both types of network was made based on the routing convergence time in the case of link failure. The Word Count and Tera Sort applications were also run to evaluate the performance of Hadoop jobs under different network topology and data sizes for data centre network.

To evaluate the routing convergence time in the case of link failure, it is proposed that the failure is occurred in any link of all possible paths specified for each shuffling flow in both topologies. The routing change of the packets in the conventional network needs some time, because any change or update in link status and routing computation has to be performed by each router in the entire network. While in SDN, the controller is the brain of the entire network management and maintains the routing process of the whole network in a centralised manner. The floodlight controller in SDN was used to manage and maintain the status of all links in the data centre network using the link layer discovery protocol (LLDP), whilst the information of the network topology was maintained by the topology service responsible for calculating the routing computation. In the conventional network, the routing module uses the flooding method to transmit the information of link status to other routers in the data centre network in a distributed manner.

Two experiments were conducted to evaluate the convergence time of the routing process. As can be seen in Figure 5.11, the convergence time of the routing process for different sizes of topology is minimised using the SDN network for the leaf-spine topology, which is not the case with the conventional network. The reason for this, is because the convergence process in the SDN network is more flexible and faster than with the conventional network. The convergence process of the latter depends on the routers, whereby each maintains a routing table which forwards and queries each packet in the network using a specific path. When any change or update occurs in the routing process of packets, like link failure, router 1 will send its update to its neighbour router 2 that will check for any required changes or updates in its routing table, then sending its update to its neighbour and so on. This is means that the changes and updates will broadcast over the whole network and consequently, it leads to slowing of convergence time in the conventional network, especially when the size of network topology is increased. This is because the routers will be scaled when the size of network is increased.

On the other hand, the floodlight controller in the SDN network is responsible for any change or update, such as link down, by using the OpenFlow control that installs flow entries into the switches. The controller can also add, delete and modify flow entries for all connected switches in the network. The SDN controller detects whether any link failure has occurred using PORT_STATUS. Furthermore, switches in the network notify the controller of any link down through error messages. When the controller receives the error messages from the connected switches, it computes new available routes based on the flow tables. As a result of the centralised manner of the SDN control, this makes the convergence routing time more rapid and agile. The Word Count and Tera Sort applications were run to evaluate the performance of a Hadoop job using the proposed system in the leaf-spine topology. The optimised values of the Hadoop parameters in chapter 3 were used in the proposed SDN network under different sizes of network topology. Moreover, different sizes of datasets ranging from 1GB to 5GB were used. In the first experiment, it can be clearly observed from Figure 5.12 that the execution time of the proposed work based on SDN is reduced when compared to the conventional network for the Word Count application. The execution time of Tera Sort application is also decreased using our proposed approach when compared to the conventional network. In the second experiment, the execution time of Word Count and Tera Sort applications is also shorter than with the conventional network as shown in Figure 5.13.

Furthermore, the execution time of both applications under 12 switches in the proposed SDN network was relatively same as that using six switches due to the centralised management of the SDN controller, which can deal with any issues of the routing process, such as congestion or link crashing, irrespective of network topology size. However, the execution time of both applications in the conventional network using 12 switches was increased when compared to utilising six. As it is mentioned above, the routing convergence time is increased when a larger network topology size is used, because of the distributed technique of the conventional network in case of congestion or link down. The dynamic routing of the scheduling and routing process based on an SDN environment has a significant impact on the performance of Hadoop jobs, which is not present in the static environment of a conventional network.

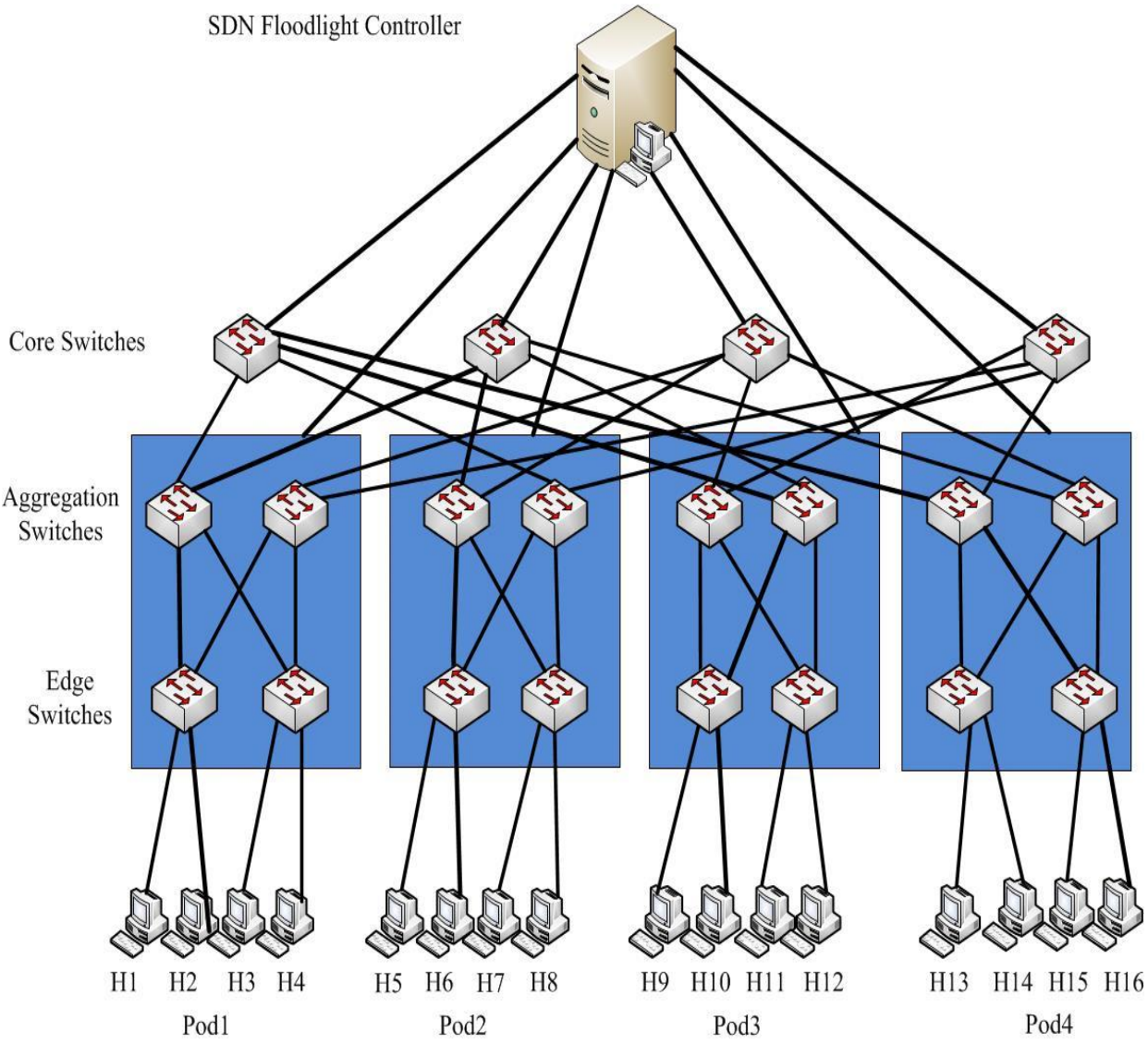

Figure 5. 7: Fat tree topology with 20 switches based on SDN

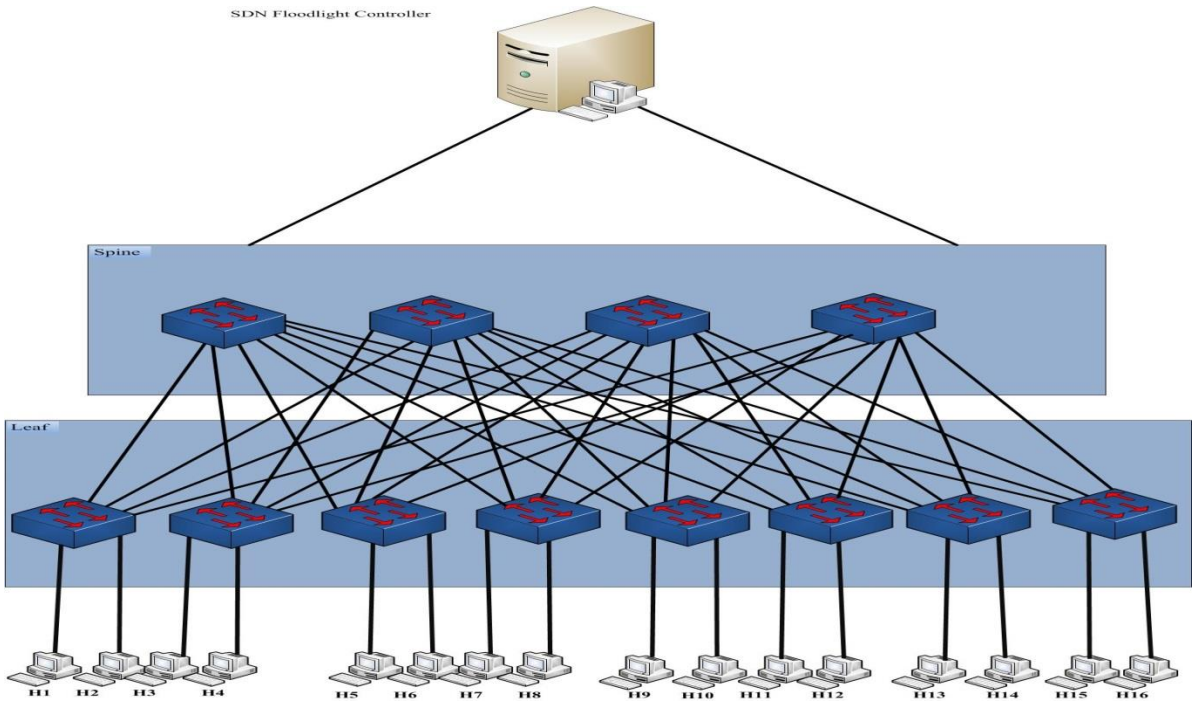

Figure 5. 8: Leaf-spine topology with 12 switches based on SDN

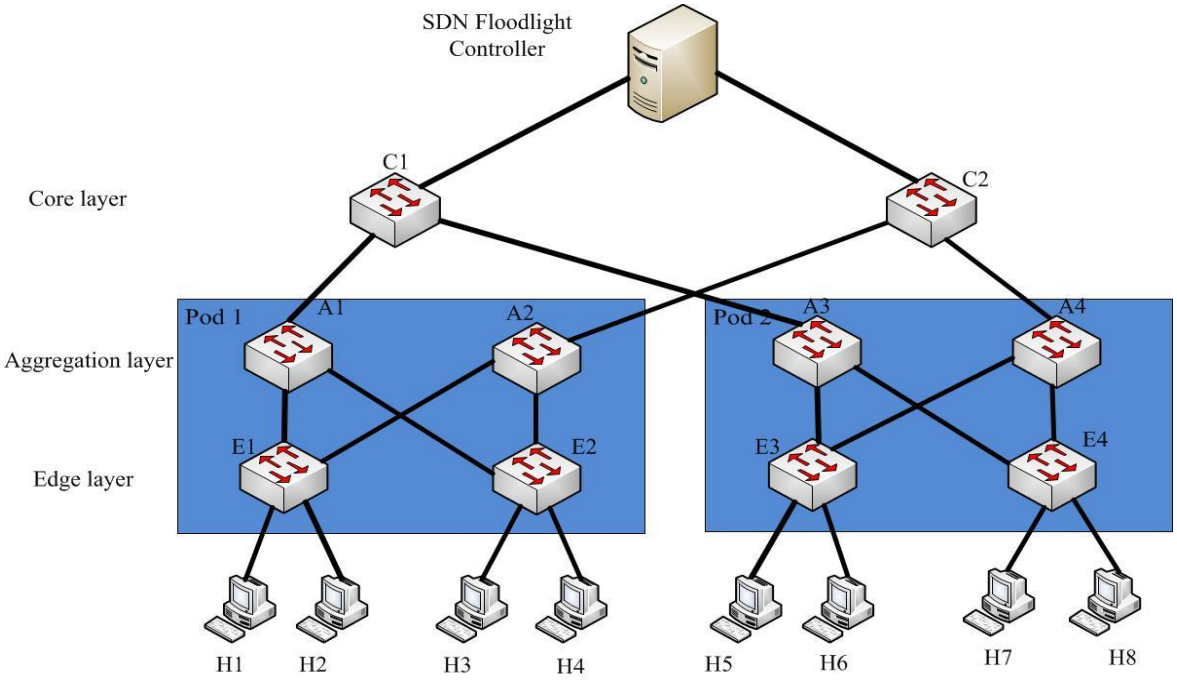

Figure 5. 9: fat tree topology with 10 switches based on SDN

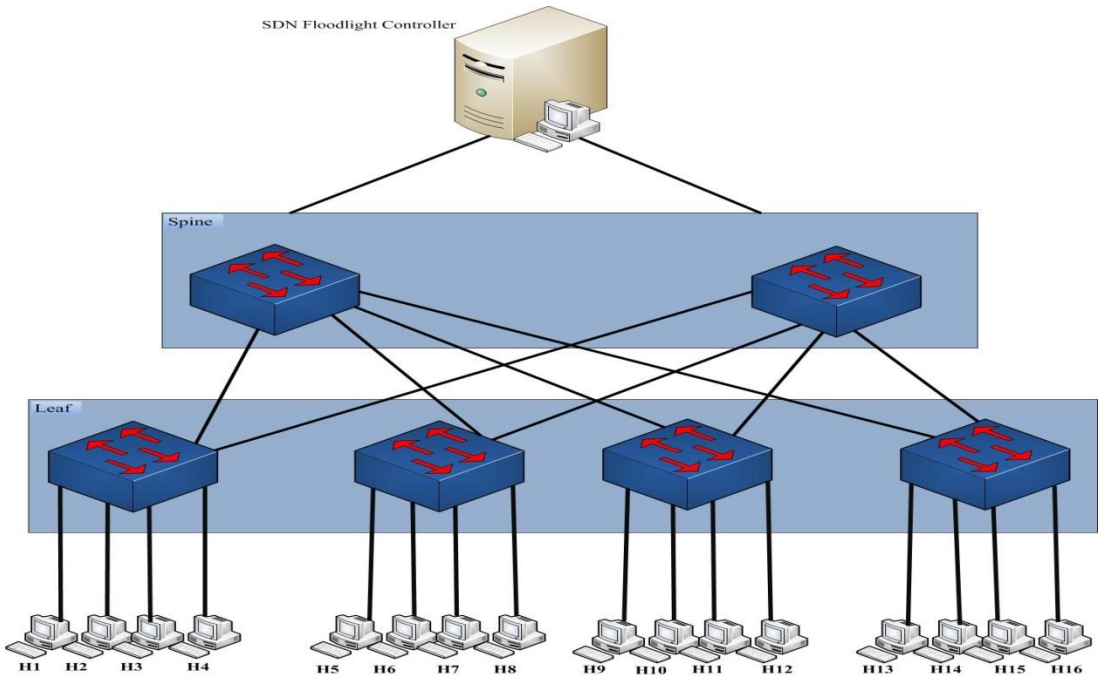

Figure 5. 10: Leaf-spine topology with 6 switches based on SDN

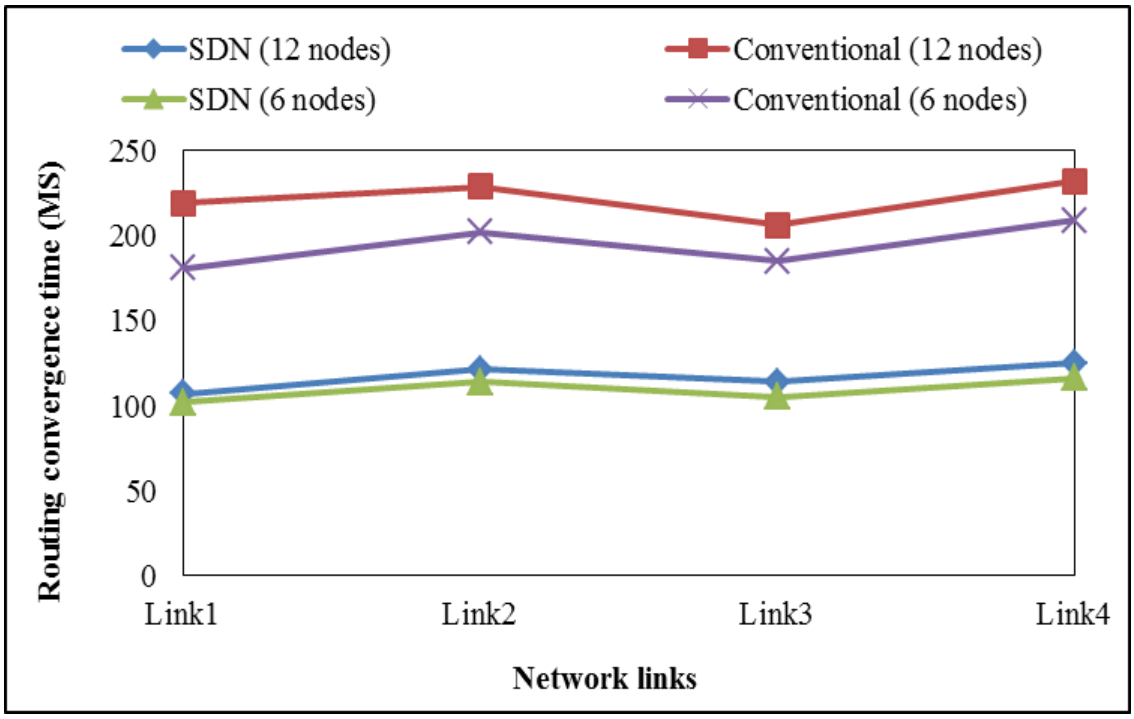

Figure 5. 11: Routing convergence time using different topology sizes

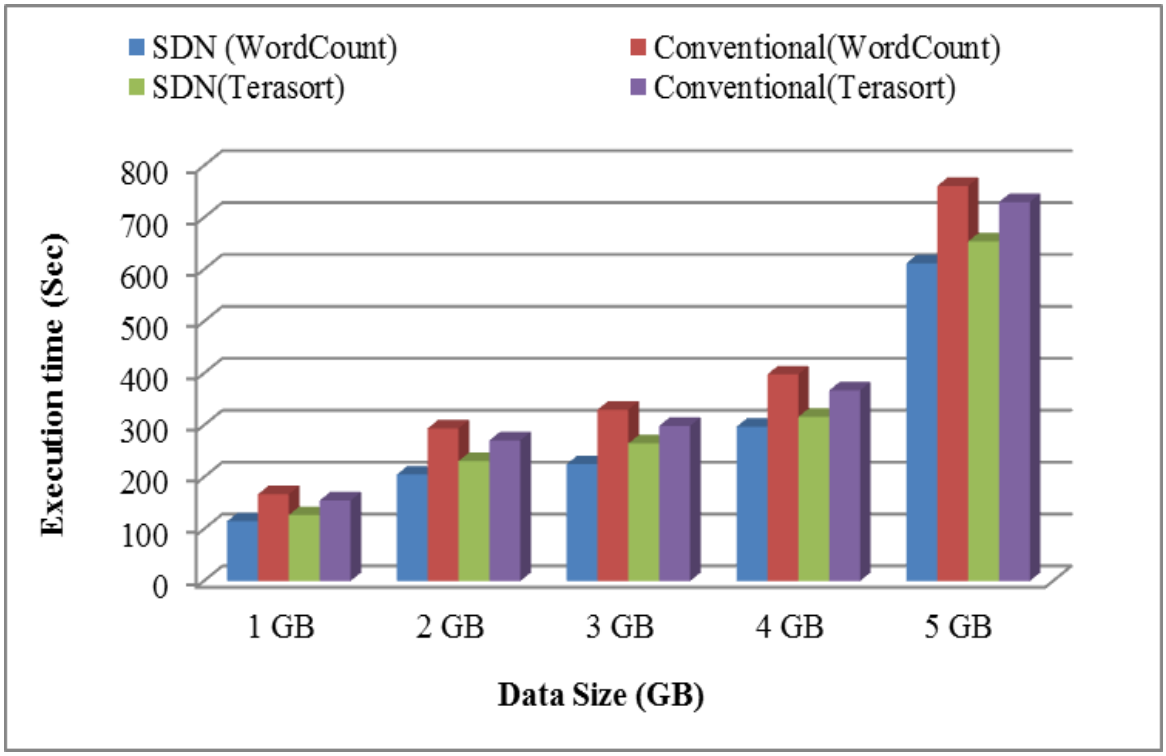

Figure 5. 12: Hadoop execution time within leaf-spine using 12 switches

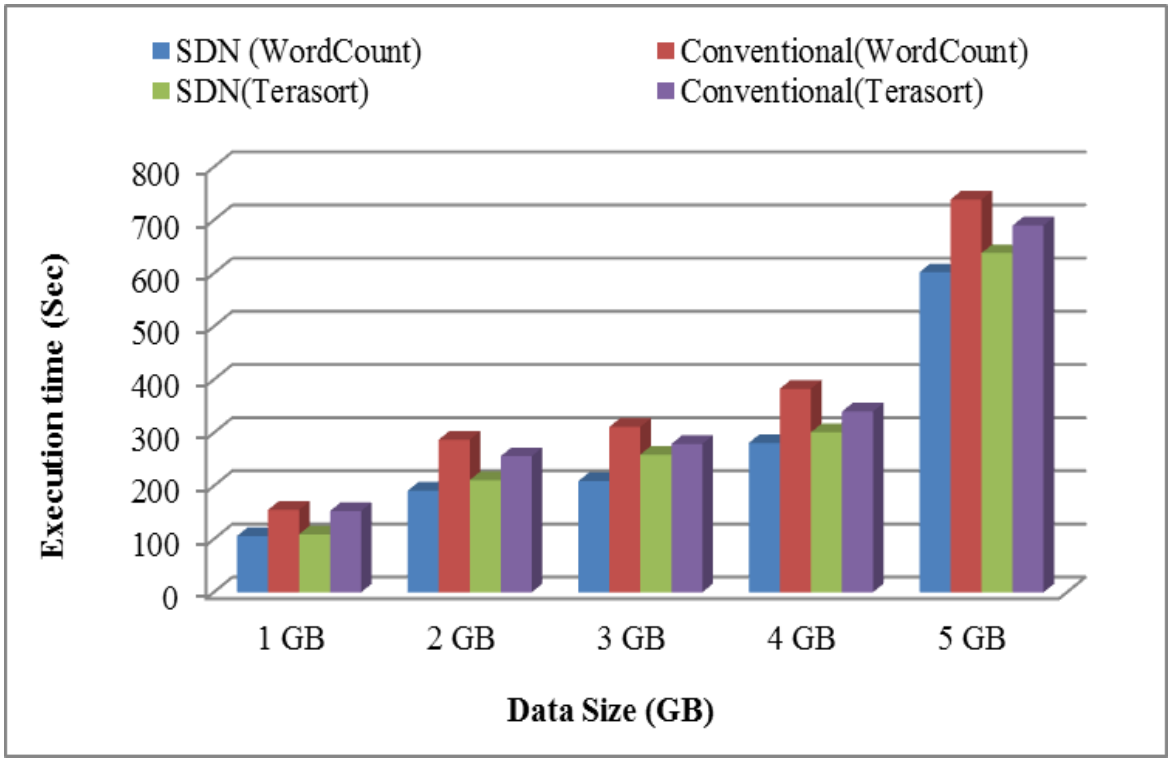

Figure 5. 13: Hadoop execution time within leaf-spine using 6 switches

5.6 Summary

This chapter proposed scheduling and routing algorithm based on SDN to achieve reliable bandwidth utilization in the leaf-spine topology of data Centre network. The proposed work consists of three modules. The first module is to monitor the current load of all links in the data centre network and obtain the statistics information from all Open vSwitch. The second one is to obtain the network resources and all required information of shuffling flows from Hadoop hosts. The routing process is performed by the scheduling and routing module, where the implementation of our scheduling algorithm. Our proposed work was evaluated by using Tera Sort and Word count applications. The results show that our proposed work outperforms ECMP and TRILL by reducing the execution time of shuffling flows in the leaf-spine topology. The leaf-spine topology was also compared with the fat tree topology and the experimental results showed that the leaf-spine has achieved better performance. Furthermore, the routing convergence time has been reduced using the SDN network for the leaf-spine topology in comparison with the conventional one under different topology sizes. The execution time of a Hadoop job was also reduced using the SDN network for the leaf-spine topology under different topology and data sizes, when compared to the conventional network.

Chapter 6

Conclusion and Future Work

The major contributions of this thesis are provided in this chapter. It also includes some considerations and proposals for future work to expand up on the work presented in this thesis.

6.1 Conclusion

In this thesis, first, an approach to automatically setting the configuration parameters of Hadoop framework, so as to improve its performance was presented. Secondly, the performance of an R and Hadoop Integrated Programming Environment for the fast calculation and analysis regarding solar irradiation datasets was evaluated. Finally, the superior performance of a Hadoop job in a large scale cluster in a data centre network was demonstrated by presenting an effective routing algorithm based on SDN technology to alleviate the produced traffic in the shuffle phase.

The first part of the thesis was focused on the optimisation of Hadoop performance by tuning the values of its parameters. The framework of Hadoop contains more than 150 parameters and some of them have a significant effect on the performance of a Hadoop job if they are tuned with the optimum values. The optimal settings of the Hadoop parameters are a challenging task as well as being time consuming. In this part, a novel technique is proposed to set the Hadoop MapReduce parameters with the optimal or near optimal settings in an automatic manner. First, Genetic programming was utilised to construct a fitness function, used to represent the interrelations among the Hadoop parameters. Second, a GA was employed to find the optimum values of the Hadoop parameters by applying the constructed fitness function.

For further improvement, SDN is employed to enhance the networking part of a Hadoop job during the shuffle phase. Furthermore, the optimised values were applied in the optimised SDN network to evaluate the job performance. The work performance was first evaluated using eight virtual machines (VMs) of a Hadoop cluster placed on a Microsoft azure cloud, whilst another cluster, consisting of fourteen VMs, was used based on SDN. The experimental results showed that the proposed approach improves

the performance of a Hadoop job in the conventional network as compared to Gunther's work and the default settings. Moreover, the performance of a Hadoop job in the SDN network is better than for the conventional one.

In the second part of the thesis, an effective computing platform that provides distributed processing and storage for large amounts of solar irradiation data was presented. This platform is built by using R language in Hadoop environment (RHIPE). RHIPE was employed for solar irradiation data analysis in a Hadoop cluster, which was evaluated in comparison with R language in terms of scalability, accuracy and speedup. Experimental results showed that RHIPE can achieve a significant improvement over the R language. Furthermore, the speedup of parallel RHIPE in the Hadoop cluster was analysed by Gustafson's Law, which was revised to improve the performance of the parallel computation on intensive irradiation data sets in the Hadoop cluster based on a cloud computing environment.

For the third part of the thesis, the focus was on the performance of a Hadoop job in a large scale cluster in a data centre network. SDN is used to improve the performance of Hadoop jobs. The networking aspect of a Hadoop cluster is optimised using SDN by achieving centralised control and agile management. This can improve the performance of a Hadoop job by accelerating the shuffling phase that can be network intensive. Two effective routing algorithms based on SDN were implemented to improve the networking part of a Hadoop job during the shuffle phase. The first algorithm was used to allocate efficient paths for each shuffling flow, according to its network resources demand as well as its size and number in the data centre network. The second algorithm was also based on SDN and used to reroute the shuffling flows to another available paths in the case of any link crashing or failure. The proposed work was evaluated using different network topologies and sizes. Both the fat tree and leaf-spine topology with different sizes were employed to assess performance, being built by EstiNet emulator software. Different sizes of network topologies were emulated consisting of 20 and 10 switches for the fat tree, whilst 12 and 6 switches were deployed for the leaf-spine. The experimental results showed the performance of a Hadoop job in the SDN network was improved when compared to the conventional network under different network topologies and sizes. Furthermore, the routing convergence time was also reduced in the case of link failure in the SDN network when compared to the conventional one.

6.2 Future Work

Despite the significant contributions of this thesis in optimising the computing and networking resources of a Hadoop framework, a number of works should be considered in the future to improve this framework's performance. Data locality algorithms could be applied to this end, which would move the computation close to the data location instead of moving the data towards the computation node. Data locality is a technique of moving the computation task near to where the actual data resides on the node. This technique is particularly effective for massive datasets, because moving the computation results in this way reduces the congestion occurring in the network and hence, improves the performance of the Hadoop system. Massive amounts of data can be transmitted across thousands of shared nodes in a cluster computing environment and this puts load on the network, thus creating congestion. However, a scheduler for Hadoop can be used to avert unnecessary data transmission and minimise network congestion so as to improve the overall throughput of the system. Data locality can be characterised into three different levels, the first being data node level locality, where the mapper is running on the same node that holds the data. In this case, the located data are very close to the computation task. In the second level, the data are located in a node and the mapper is running on another node within the same rack, which is called rack level locality. The third level, termed inter-rack level, is where the mapper is running on a node and the data are located on another in a different rack.

Furthermore, SDN can be used in a Hadoop cluster to improve data locality by providing bandwidth-aware scheduling since the network bandwidth is a scare resource. Such a system can allocate tasks in a global view and assign them effectively in an optimised way using SDN technology. This provides agile control and centralised management through the separation of the control plane from the data plane.

In this thesis, RHIPE was utilised for analysing solar irradiation datasets with this process being conducted offline using estimated data provided by the London Weather Centre for the period 1996-2005. In future, an online collection method for solar irradiation data could be used to process them in real time using a Hadoop framework. Furthermore, SDN can be applied for RHIPE for better optimisation.

Finally, in this thesis the computing and networking performance of a Hadoop MapReduce job in a homogenous cluster where all the computing nodes are

homogenous was evaluated. Future work could be aimed at determining the networking and computing performance of Hadoop in a heterogeneous cluster, where the nodes have different computing capacity.

References

[1] "Big data and cloud computing – challenges and opportunities." [Online]. Available: http://bigdata-madesimple.com/big-data-and-cloud-computing-challenges-and-opportunities/. [Accessed: 11-Apr-2018].

[2] "What Is Big Data? | SAS UK." [Online]. Available: https://www.sas.com/en_gb/insights/big-data/what-is-big-data.html. [Accessed: 11-Apr-2018].

[3] "Big Data - The 5 Vs Everyone Must Know." [Online]. Available: https://www.slideshare.net/BernardMarr/140228-big-data-volume-velocity-variety-varacity-value/2-To_get_a_better_understanding. [Accessed: 11-Apr-2018].

[4] "What is machine-to-machine (M2M)? - Definition from WhatIs.com." [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/machine-to-machine-M2M. [Accessed: 11-Apr-2018].

[5] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything," *CISCO white Pap.*, no. April, pp. 1–11, 2011.

[6] "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016." [Online]. Available: https://www.gartner.com/newsroom/id/3598917. [Accessed: 11-Apr-2018].

[7] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," in *3rd Nirma University International Conference on Engineering, NUiCONE 2012,* 2012.

[8] L. Wang, M. Kunze, J. Tao, and G. Von Laszewski, "Towards building a cloud for scientific applications," *Adv. Eng. Softw.*, vol. 42, no. 9, pp. 714–722, 2011.

[9] B. B. P. Rao, P. Saluia, N. Sharma, a Mittal, and S. V Sharma, "Cloud computing for Internet of Things & sensing based applications," *Sens. Technol. (ICST), 2012 Sixth Int. Conf.*, pp. 374–380, 2012.

[10] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-Peer

Cloud Provisioning: Service Discovery and Load-Balancing," no. i, pp. 195–217, 2010.

[11] L. Wang *et al.*, "Cloud computing: A perspective study," *New Gener. Comput.*, vol. 28, no. 2, pp. 137–146, 2010.

[12] L. Wang *et al.*, "MapReduce across distributed clusters for data-intensive applications," *Proc. 2012 IEEE 26th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2012*, pp. 2004–2011, 2012.

[13] "Welcome to Apache™ Hadoop®! ali." [Online]. Available: http://hadoop.apache.org/. [Accessed: 12-Apr-2018].

[14] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," *BMC Bioinformatics*, vol. 11, no. SUPPL. 12, pp. 1–6, 2010.

[15] "Welcome to Apache™ Hadoop®!" [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Sep-2017].

[16] "PoweredBy - Hadoop Wiki." [Online]. Available: https://wiki.apache.org/hadoop/PoweredBy. [Accessed: 13-Mar-2018].

[17] S. G. Manikandan, "Big Data Analysis using Apache Hadoop."

[18] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.

[19] "Hadoop - Big Data Overview." [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_big_data_overview.htm. [Accessed: 27-Oct-2018].

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," *2010 IEEE 26th Symp. Mass Storage Syst. Technol. MSST2010*, pp. 1–10, 2010.

[21] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Proj. Website*, no. August, pp. 1–14, 2007.

[22] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *Proc.*

Ninet. ACM Symp. Oper. Syst. Princ. - SOSP '03, p. 29, 2003.

[23] Wikipedia, "Google File System - Wikipédia." p. 1, 2017.

[24] Hortonworks, "Apache Hadoop HDFS - Hortonworks." 2017.

[25] D. Desani, V. Gil-Costa, C. A. C. Marcondes, and H. Senger, "Black-Box Optimization of Hadoop Parameters Using Derivative-Free Optimization," *Proc. - 24th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2016*, pp. 43–50, 2016.

[26] B. Andrea and A. Bennedeto, "Nuevas alternativas para pensar el desarrollo de los territorios rurales . Posibilidades y riesgos 1," *Cuad. Desarro. Rural*, vol. 57, no. 57, pp. 101–131, 2006.

[27] L. Cai, Y. Qi, and J. Li, "A Recommendation-Based Parameter Tuning Approach for Hadoop," *2017 IEEE 7th Int. Symp. Cloud Serv. Comput.*, pp. 223–230, 2017.

[28] B. Palanisamy, A. Singh, L. Liu, and B. Langston, "Cura: A cost-optimized model for MapReduce in a cloud," *Proc. - IEEE 27th Int. Parallel Distrib. Process. Symp. IPDPS 2013*, pp. 1275–1286, 2013.

[29] V. Jalaparti, H. Ballani, and P. Costa, "Bazaar: Enabling Predictable Performance in Datacenters," *Research.Microsoft.Com*, 2012.

[30] S. Babu, "Towards automatic optimization of MapReduce programs," *SoCC '10 Proc. 1st ACM Symp. Cloud Comput.*, pp. 137–142, 2010.

[31] J. Yan, X. Yang, R. Gu, C. Yuan, and Y. Huang, "Performance optimization for short MapReduce job execution in Hadoop," *Proc. - 2nd Int. Conf. Cloud Green Comput. 2nd Int. Conf. Soc. Comput. Its Appl. CGC/SCA 2012*, pp. 688–694, 2012.

[32] K. Elmeleegy, "Piranha: Optimizing Short Jobs in Hadoop," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 985–996, 2013.

[33] R. Gu *et al.*, "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2166–2179, 2014.

[34] “Top 6 Hadoop Vendors providing Big Data Solutions in Open Data Platform.” [Online]. Available: https://www.dezyre.com/article/top-6-hadoop-vendors-providing-big-data-solutions-in-open-data-platform/93. [Accessed: 28-Mar-2018].

[35] “hadoop - What is Hue all about? - Stack Overflow.” [Online]. Available: https://stackoverflow.com/questions/24115828/what-is-hue-all-about. [Accessed: 02-Nov-2018].

[36] “Hue, the self-service open source Analytics Workbench for browsing, querying and visualizing data interactively | Hue is an open source Query Tool for browsing, querying and visualizing data with focus on SQL and Search.” [Online]. Available: http://gethue.com/. [Accessed: 02-Nov-2018].

[37] “Apache Hive TM.” [Online]. Available: https://hive.apache.org/. [Accessed: 02-Nov-2018].

[38] “Hadoop Ecosystem and their Components - A complete Tutorial - DataFlair.” .

[39] Yah, “Apache Pig.” 2008.

[40] “APache Oozie Introduction.” [Online]. Available: https://www.tutorialspoint.com/apache_oozie/apache_oozie_introduction.htm. [Accessed: 26-Mar-2018].

[41] “Oozie -.” [Online]. Available: https://oozie.apache.org/docs/4.0.0/DG_Overview.html. [Accessed: 26-Mar-2018].

[42] F. Chang, J. Dean, and S. Ghemawat, “, ‘Bigtable: A Distributed Storage System for Structured Data,’” *ACM Trans. Comput. Syst. Vol. 26, No.*, vol. 2, no. 2, pp. 1–426, 2008.

[43] “Hadoop Ecosystem and Components - BMC Software.” [Online]. Available: http://www.bmcsoftware.uk/guides/hadoop-ecosystem.html. [Accessed: 02-Nov-2018].

[44] “HBase Tutorial.” [Online]. Available: https://www.tutorialspoint.com/hbase/. [Accessed: 02-Nov-2018].

[45] "Apache ZooKeeper - Home." [Online]. Available: https://zookeeper.apache.org/. [Accessed: 02-Nov-2018].

[46] "HBase Architecture phylips @ bmy." 2011.

[47] "Apache Flume - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Apache_Flume. [Accessed: 25-Mar-2018].

[48] "Apache Flume Architecture." [Online]. Available: https://www.tutorialspoint.com/apache_flume/apache_flume_architecture.htm. [Accessed: 25-Mar-2018].

[49] "Welcome to Apache Flume — Apache Flume." [Online]. Available: https://flume.apache.org/. [Accessed: 25-Mar-2018].

[50] "Apache Sqoop - Hortonworks." [Online]. Available: https://hortonworks.com/apache/sqoop/. [Accessed: 25-Mar-2018].

[51] "Sqoop Tutorial." [Online]. Available: https://www.tutorialspoint.com/sqoop/index.htm. [Accessed: 25-Mar-2018].

[52] "hadoop2 - Differnce between Hadoop 1 and Hadoop 2 - Stack Overflow." [Online]. Available: https://stackoverflow.com/questions/24993570/differnce-between-hadoop-1-and-hadoop-2. [Accessed: 29-Mar-2018].

[53] "10 Big Differences between Hadoop1 and Hadoop2 | Acadgild Blog." [Online]. Available: https://acadgild.com/blog/10-big-differences-between-hadoop1-and-hadoop2/. [Accessed: 29-Mar-2018].

[54] "Apache Hadoop 2.4.1 - Hadoop Distributed File System-2.4.1 - Federation." [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/Federation.html. [Accessed: 29-Mar-2018].

[55] "Hadoop – Apache Hadoop 3.0.0." [Online]. Available: https://hadoop.apache.org/docs/r3.0.0/. [Accessed: 29-Mar-2018].

[56] "Comparison Between Hadoop 2.x vs Hadoop 3.x - DataFlair." [Online]. Available: https://data-flair.training/blogs/hadoop-2-x-vs-hadoop-3-x-comparison/. [Accessed: 29-Mar-2018].

[57] “How Apache Hadoop 3 Adds Value Over Apache Hadoop 2 - Hortonworks.” [Online]. Available: https://hortonworks.com/blog/hadoop-3-adds-value-hadoop-2/. [Accessed: 29-Mar-2018].

[58] “Mahout Introduction.” [Online]. Available: https://www.tutorialspoint.com/mahout/mahout_introduction.htm. [Accessed: 26-Mar-2018].

[59] “R: What is R?” [Online]. Available: https://www.r-project.org/about.html. [Accessed: 27-Mar-2018].

[60] “R and Hadoop Data Analysis – RHadoop | BigHadoop.” [Online]. Available: https://bighadoop.wordpress.com/2013/02/25/r-and-hadoop-data-analysis-rhadoop/. [Accessed: 28-Mar-2018].

[61] T. Subject *et al.*, *Big Data Analytics with R and Hadoop*. Packt Publishing, Birmingham - Mumbai, 2015.

[62] J. Rounds, “RHIPE - R and Hadoop Integrated Processing Environment project home page.”

[63] K. Bilal, S. Khan, J. Kolodziej, and L. Zhang, “A Comparative Study Of Data Center Network Architectures.,” *Proceeding 26th Eur. Conf. Model. Simul.*, vol. 0, no. Cd, pp. 1–7, 2012.

[64] Y. Liu, J. K. Muppala, and M. Veeraraghavan, “A survey of data center network architectures,” *Ieee Commun. Surv. Tutorials*, vol. 15, no. 2, pp. 1–22, 2014.

[65] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, p. 63, 2008.

[66] Cisco Systems, “Data Center Architecture Overview,” pp. 1–10, 2014.

[67] “Data center network architectures - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Data_center_network_architectures. [Accessed: 30-Mar-2018].

[68] L. W. Cheng and S. Y. Wang, “Application-aware SDN routing for big data networking,” *2015 IEEE Glob. Commun. Conf. GLOBECOM 2015*, 2015.

[69] M. Brown, “Front cover Introduction to Grid,” *Contract*, no. November, 2017.

[70] “OSPF Design Guide - Cisco.” [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html. [Accessed: 29-Oct-2018].

[71] C. E. H. <chopps@merit.edu>, “Analysis of an Equal-Cost Multi-Path Algorithm.”

[72] L. Romero, “‘Cloud computing’ in library automation. Benefits and drawbacks.,” *Bottom Line Manag. Libr. Finan.*, vol. 25, no. 3, pp. 110–115, 2012.

[73] J. Shayan, A. Azarnik, S. Chuprat, and S. Karamizadeh, “Identifying Benefits and risks associated with utilizing cloud computing,” *Int. J. Soft Comput. Softw. Eng.*, vol. 3, no. 3, pp. 416–421, 2013.

[74] S. Turner, “Benefits and risks of cloud computing,” *J. Technol. Res.*, vol. 4, pp. 1–7, 2013.

[75] “Cloud Computing Overview.” [Online]. Available: https://www.tutorialspoint.com/microsoft_azure/cloud_computing_overview.htm. [Accessed: 02-Apr-2018].

[76] D. Wu *et al.*, “CLOUD COMPUTING – An Overview An Overview,” *White Pap.*, vol. 462, no. 7276, pp. 1–5, 2009.

[77] M. Anandarajan and B. Arinze, “Cloud Computing,” *Wiley Encycl. Manag.*, pp. 1–27, 2015.

[78] “Software-Defined Networking (SDN) Definition - Open Networking Foundation.” [Online]. Available: https://www.opennetworking.org/sdn-definition/. [Accessed: 02-Apr-2018].

[79] O.N.F., “Software-defined networking: The new norm for networks,” *ONF White Pap.*, vol. 2, pp. 2–6, 2012.

[80] W. Braun and M. Menth, “Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,” *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[81] “Understanding the SDN Architecture and SDN Control Plane - Defini.” [Online]. Available: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/. [Accessed: 03-Apr-2018].

[82] C. Fernandez and J. L.Muñoz, “Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu,” p. 183.

[83] “Floodlight OpenFlow Controller -Project Floodlight.” [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed: 03-Apr-2018].

[84] L. V. Morales, A. F. Murillo, and S. J. Rueda, “Extending the floodlight controller,” *Proc. - 2015 IEEE 14th Int. Symp. Netw. Comput. Appl. NCA 2015*, pp. 126–133, 2016.

[85] P. Qin, B. Dai, B. Huang, and G. Xu, “Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data,” *IEEE Syst. J.*, pp. 1–8, 2015.

[86] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, “BAR: An efficient data locality driven task scheduling algorithm for cloud computing,” *Proc. - 11th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGrid 2011*, pp. 295–304, 2011.

[87] S. Zhao and D. Medhi, “Application-Aware Network Design for Hadoop MapReduce Optimization Using Software-Defined Networking,” *IEEE Trans. Netw. Serv. Manag.*, vol. 4537, no. c, pp. 1–14, 2017.

[88] A. Pavlo *et al.*, “A comparison of approaches to large-scale data analysis,” *SIGMOD-PODS’09 - Proc. Int. Conf. Manag. Data 28th Symp. Princ. Database Syst.*, pp. 165–178, 2009.

[89] “7 Tips for Improving MapReduce Performance – Cloudera Engineering Blog.” [Online]. Available: http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/. [Accessed: 15-Aug-2017].

[90] D. Wu and A. Gokhale, “A Self-Tuning System based on Application Profiling and Performance Analysis for Optimizing Hadoop MapReduce Cluster Configuration.”

[91] G. Liao, K. Datta, and T. L. Willke, “Gunther: Search-based auto-tuning of MapReduce,” in *Lecture Notes in Computer Science (including subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013.

[92] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir, “Panacea: Towards Holistic Optimization of MapReduce Applications.”

[93] Y. Li *et al.*, “Breaking the boundary for whole-system performance optimization of big data,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2013.

[94] Impetus, “Hadoop Performance Tuning,” *White Pap.*, no. October 2009, pp. 1–13, 2009.

[95] T. White, *Hadoop : the definitive guide*. O’Reilly, 2010.

[96] M. Li, L. Zeng, and S. Meng, “MR O NLINE : MapReduce Online Performance Tuning,” *Hpdc*, pp. 165–176, 2014.

[97] H. Herodotou and S. Babu, “Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs,” *PVLDB Proc. VLDB Endow.*, vol. 4, no. 11, pp. 1111–1122, 2011.

[98] H. Herodotou, H. Lim, G. Luo, N. Borisov, and L. Dong, “Starfish : A Self-tuning System for Big Data Analytics,” *Cidr*, vol. 11, pp. 261–272, 2011.

[99] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programing*, no. March. 2008.

[100] M. Walker, “Introduction to genetic programming,” *Tech. Np Univ. Mont.*, pp. 1–9, 2001.

[101] J. McCall, “Genetic algorithms for modelling and optimisation,” *J. Comput. Appl. Math.*, vol. 184, no. 1, pp. 205–222, 2005.

[102] “Genetic Algorithms Introduction.” [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm. [Accessed: 20-Apr-2018].

[103] Open Networking Foundation, “SDN Architecture Overview,” *Onf*, pp. 1–5, 2013.

[104] R. Gowri, “MR-GABiT : Map Reduce based Genetic Algorithm for Biclustering Time Series Data,” pp. 381–387, 2016.

[105] G. Reinert, “Time Series,” *Hilar. Term, USA*, no. 1990, pp. 1–66, 2010.

[106] C. Severance and K. Dowd, “High Performance Computing,” *Hpc*, p. 294, 2012.

[107] X. Wang, Z. Yan, and L. Li, “A grid computing based approach for the power system dynamic security assessment,” *Comput. Electr. Eng.*, vol. 36, no. 3, pp. 553–564, 2010.

[108] G. A. Ezhilarasi and K. Swarup, “Parallel contingency analysis in a high performance computing environment,” *Int. Conf. Power Syst. 2009.*, pp. 1–6, 2009.

[109] E. Gabriel *et al.*, “Open MPI: Goals, concept, and design of a next generation MPI implementation,” *11th Eur. PVM/MPI Users’ Gr. Meet.*, pp. 97–104, 2004.

[110] I. Gorton *et al.*, “A High-performance hybrid computing approach to massive contingency analysis in the power grid,” *e-Science 2009 - 5th IEEE Int. Conf. e-Science*, pp. 277–283, 2009.

[111] W. Gao and X. Chen, “Distributed Generation Placement Design and Contingency Analysis with Parallel Computing Technology,” *Fuel*, vol. 4, no. 4, pp. 347–354, 2009.

[112] J. Yin, Y. Liao, M. Baldi, L. Gao, and A. Nucci, “GOM-Hadoop: A distributed framework for efficient analytics on ordered datasets,” *J. Parallel Distrib. Comput.*, vol. 83, pp. 58–69, 2015.

[113] J. Li, X. Ma, S. Yoginath, G. Kora, and N. F. Samatova, “Transparent runtime parallelization of the R scripting language,” *J. Parallel Distrib. Comput.*, vol. 71, no. 2, pp. 157–168, 2011.

[114] B. Agrawal, A. Chakravorty, C. Rong, and T. W. Wlodarczyk, “R2Time: A framework to analyse open TSDB Time-series data in HBase,” in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2015.

[115] L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro, “A parallel genetic algorithm based on hadoop MapReduce for the automatic generation of junit test suites,” in *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, 2012.

[116] L. Li, F. Noorian, D. J. M. Moss, and P. H. W. Leong, “Rolling Window Time Series Prediction using MapReduce.”

[117] Y. Y. Liu, P. Thulasiraman, and R. K. Thulasiram, “Parallelizing Active Memory Ants with MapReduce for Clustering Financial Time Series Data,” in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016.

[118] C. Sheng, J. Zhao, H. Leung, and W. Wang, “Extended Kalman Filter based Echo State Network for 7ime Series Prediction using MapReduceFramework.”

[119] F. Bach, H. K. Çakmak, H. Maass, and U. Kuehnapfel, “Power grid time series data analysis with pig on a Hadoop cluster compared to multi core systems,” *Proc. 2013 21st Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2013*, pp. 208–212, 2013.

[120] J. D. Brutlag, “14th Systems Administration Conference Aberrant Behavior Detection in Time Series for Network Monitoring,” *Proc. 14th Syst. Adm. Conf. (LISA 2000)*, no. 14, pp. 139–146, 2000.

[121] H. Yin, S. Yang, S. Ma, F. Liu, and Z. Chen, “A novel parallel scheme for fast similarity search in large time series,” *China Commun.*, vol. 12, no. 2, pp. 129–140, 2015.

[122] P. Laurinec, M. Lóderer, P. Vrablecová, M. Lucká, V. Rozinajová, and A. B. Ezzeddine, “Adaptive Time Series Forecasting of Energy Consumption using Optimized Cluster Analysis,” 2016.

[123] K. Yang and T. Koike, “A general model to estimate hourly and daily solar radiation for hydrological studies,” *Water Resour. Res.*, vol. 41, no. 10, pp. 1–13, 2005.

[124] U. Dokdmm *et al.*, “A Predictive Evaluation of Global Solar Radiation using Recurrent Neural Models and Weather Data,” vol. 5, pp. 1–5.

[125] R. Langella, D. Proto, and A. Testa, “A Statistical Model of Solar Radiation Daily Variability,” *2014 Int. Conf. Probabilistic Methods Appl. to Power Syst.*, pp. 1–6, 2014.

[126] J. A. A and L. B. B, “Analysis of Statistical Methods for Estimating Solar Radiation,” vol. 18, no. 1, pp. 1–5, 2014.

[127] E. Yilmaz, B. Cancino, and W. R. Parra, “Statistical analysis of solar radiation data,” *Energy Sources, Part A Recover. Util. Environ. Eff.*, vol. 29, no. 15, pp. 1371–1383, 2007.

[128] O. Perpiñán, L. Maintainer, and O. P. Lamigueiro, “Title Radiation and Photovoltaic Systems,” 2016.

[129] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks,” *ACM SIGOPS Oper. Syst. Rev.*, pp. 59–72, 2007.

[130] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars: A MapReduce Framework on Graphics Processors,” *Proc. 17th Int. Conf. Parallel Archit. Compil. Tech. - PACT ’08*, p. 260, 2008.

[131] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for multi-core and multiprocessor systems,” *Proc. - Int. Symp. High-Performance Comput. Archit.*, pp. 13–24, 2007.

[132] “Welcome to Apache™ Hadoop®!” [Online]. Available: http://hadoop.apache.org/. [Accessed: 13-Mar-2018].

[133] X. H. Sun and Y. Chen, “Reevaluating Amdahl’s law in the multicore era,” *J. Parallel Distrib. Comput.*, vol. 70, no. 2, pp. 183–188, 2010.

[134] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Http://Www.Cs.Berkeley.Edu/~Jtma/Papers/Orchestra-Sigcomm2011.Pdf,” *Cs.Berkeley.Edu*.

[135] S. Narayan, S. Bailey, and A. Daga, “Hadoop acceleration in an openflow-based cluster,” *Proc. - 2012 SC Companion High Perform. Comput. Netw. Storage Anal. SCC 2012*, pp. 535–538, 2012.

[136] Anupam Das; Cristian Lumezanu; Yueping Zhang; Vishal Singh; Guofei Jiang; Curtis Yu, “Transparent and Flexible Network Management for Big Data Processing in the Cloud,” *USENIX Work. Hot Top. Cloud Comput.*, 2013.

[137] C. Y. Lin and J. Y. Liao, “An SDN app for hadoop clusters,” *Proc. - IEEE 7th Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2015*, pp. 458–461, 2016.

[138] H. Hu, Y. Wen, Y. Gao, T. S. Chua, and X. Li, “Toward an SDN-enabled big data platform for social TV analytics,” *IEEE Netw.*, vol. 29, no. 5, pp. 43–49, 2015.

[139] M. Al-Fares, S. Radhakrishnan, and B. Raghavan, “Hedera: Dynamic flow scheduling for data center networks,” *Proc. 7th*, 2010.

[140] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, “Scalable, optimal flow routing in datacenters via local link balancing,” *Proc. ninth ACM Conf. Emerg. Netw. Exp. Technol. - Conex. ’13*, pp. 151–162, 2013.

[141] F. Ieee *et al.*, “Software-Defined Networking : A Comprehensive Survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[142] N. McKeown *et al.*, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.

[143] “Fat-Tree Design | ClusterDesign.org.” [Online]. Available: https://clusterdesign.org/fat-trees/. [Accessed: 19-Jul-2018].

[144] “Understanding Spanning Tree Protocol Used for Eliminating Bridge Loops in Ethernet LANs - Technical Documentation - Support - Juniper Networks.” [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/spanning-tree-protocol-mx-ex-series-overview.html. [Accessed: 19-Jul-2018].

[145] “A Beginner’s Guide to Leaf-Spine Network Topology.” [Online]. Available: https://blog.westmonroepartners.com/a-beginners-guide-to-understanding-the-

leaf-spine-network-topology/. [Accessed: 19-Jul-2018].

[146] “LinkLayerDiscoveryProtocol - The Wireshark Wiki.” [Online]. Available: https://wiki.wireshark.org/LinkLayerDiscoveryProtocol. [Accessed: 23-Feb-2018].