Technical University of Berlin (TU Berlin)

Master Thesis

An Analysis of Side-effects of Hadoop Optimizations in the Data-center

Author:
Khwaja Zubair Sediqi

Supervisors:
Prof. Anja Feldmann
Prof. Kao Odej
Dr. Marco Canini
Lalith Surish

A thesis submitted in fulfilment of the requirements
for the degree of Master of Computer Science

in the

Intelligent Network Group(FG INET)
Department of Telecommunication Systems

October 2013

Declaration of Authorship

I, Khwaja Zubair Sediqi, declare that this thesis titled, 'An Analysis of Side-effects of Hadoop Optimizations in the Data-center' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, "

Khwaja Zubair Sediqi

TECHNICAL UNIVERSITY OF BERLIN (TU BERLIN)

Abstract

Faculty of Electrical Engineering and Computer Science
Department of Telecommunication Systems

Master of Computer Science

An Analysis of Side-effects of Hadoop Optimizations in the Data-center

by Khwaja Zubair Sediqi

The increasing volume of data from user interaction to Internet-based applications requires new storage and processing technologies. Typical databases can not process large datasets of terabytes in reasonable time. MapReduce is a programming paradigm that was invented by Yahoo! to process large datasets in key/value pattern. Hadoop is an open source implementation of MapReduce, that is used to store and process large datasets in parallel and distributed manner on cluster of commodity computers. Since preparing and maintenance of infrastructure for storage and process of large datasets requires infrastructure, storage and computational resources, which costs expensive to organizations. As solution, organizations can use cloud computing services, to store, retrieve and process large datasets and use provided resources according to their need.

Running large clusters of Hadoop on cloud infrastructure is expensive for organizations. The optimization added to Hadoop schedulers provides the opportunity of multi-tenancy for the organization, where sing Hadoop cluster can be shared by multiple organization. Cloud operators such as AMAZON EC2, provides virtualized infrastructure, where virtual machines are collocated on single physical machine. This thesis is about analysis of side effects of collocated VMs (Collocated VMs are datanodes of Hadoop cluster(s)) on performance of Hadoop. The thesis also evaluates the optimizations made to Hadoop schedulers. More precisely, the performance of fairshare scheduler and capacity share schedulers are analysed. The effect of speculative task execution for mentioned schedulers is also explained.

As result, we want to show that speculative task execution for small jobs does not improve the performance of Hadoop performance. In one hand, the evaluation shows that performance of capacity share scheduler is better than fairshare scheduler. On the other hand, capacity share scheduler has worst fairness for job completion time in comparison to fairshare scheduler. We also proof that for all the schedulers the collocation of datanodes VMs from same cluster has better performance results in comparison to collocation of VMs from different clusters. The experiment results show, though collocation of datanodes has negative impact by increasing job completion time for both capacity share and fairshare schedulers of Hadoop, but capacity share scheduler is more robust to collocation of datanodes.

Acknowledgements

I begin these acknowledgements by expressing my gratitude to my supervisors Prof. Anja Feldmann and Prof. Kao Odej who guided me through this research and were great source of encouragement.

I would like to express my gratitude to my advisor Dr. Marco Canini, that gave me freedom to pursue my own ideas and directions, while also providing me valuable feedback and guidance throughout this work. I would like to thank my advisor Mr. Lalith Suresh, for supporting me in all phases of this thesis including setup of testbed, running experiments and helping me to solve various problems. I would also like to thank members of FG INET group for the availability and support with the clusters during the evaluation phase of this work.

Furthermore, I want to thank ZiiK/TU Berlin and DAAD for their support and scholarship I benefited during this master's program. I want to thank all my friends and relatives for the encouragement and support during this master's program.

Last, but not the least, I would like to thank my family for their immense support, love and encouragement, without whom none of this would have been possible.

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements v

List of Figures viii

Abbreviations ix

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Objective . . . 3
- 1.3 Contribution . . . 3
- 1.4 Thesis Overview . . . 4

2 Background 5
- 2.1 MapReduce . . . 5
- 2.2 Hadoop . . . 7
  - 2.2.1 HDFS . . . 8
    - 2.2.1.1 HDFS Blocks . . . 9
  - 2.2.2 Namenode and Datanode . . . 9
  - 2.2.3 Schedulers . . . 10
- 2.3 MapReduce Versions . . . 11
  - 2.3.0.1 MapReduce Version 1 . . . 11
  - 2.3.0.2 MapReduce version 2 (Yarn) . . . 12
- 2.4 Cloud Computing . . . 14
  - 2.4.1 Service Models . . . 14
  - 2.4.2 Deployment Models . . . 15

3 Hadoop Optimizations 16
- 3.1 Capacity Scheduler . . . 16
- 3.2 Fairshare Scheduler . . . 18
- 3.3 Speculative Execution . . . 19
- 3.4 LATE Scheduler . . . 20

3.4.1 Advantages of LATE 22

4 Methodology **23**
4.1 Objective of The Work 23
4.2 Methodology 24
4.3 Work Scope 25
4.4 Experimental Environment 26
4.4.1 Physical Resources 27
4.4.2 Terasort 27
4.4.3 OpenStack 28
4.4.4 Ubuntu Virtual Machine 28

5 Evaluation **29**
5.1 Experiment 1 30
5.1.1 Objectives 30
5.1.2 Results 30
5.1.3 Fairness 33
5.1.4 Discussion 34
5.2 Experiment 2 34
5.2.1 Objectives 34
5.2.2 Results 35
5.2.3 Fairness 37
5.2.4 Discussion 37
5.3 Experiment 3 39
5.3.1 Objective 39
5.3.2 Results 39
5.3.3 Fairness 42
5.3.4 Discussion 42

6 Cloud Computing in Afghanistan **44**
6.1 Afghanistan and ICT 44
6.1.1 National Optical Fiber Backbone 45
6.2 Challenges 45
6.3 Cloud Computing In Afghanistan 46

7 Conclusion **47**
7.1 Conclusions 47
7.2 Future Work 48

Bibliography **49**

List of Figures

2.1 MapReduce logical data flow [7] 6
2.2 MapReduce Version 1 Architecture [7] 12
2.3 YARN architecture [8] 13

5.1 Experiment 1: Mean values of schedulers' performance 31
5.2 Experiment 1: Fairness comparison of schedulers' performance 33
5.3 Mean value of schedulers' performance 35
5.4 Experiment 2: Fairness comparison of schedulers' performance 38
5.5 Experiment 3: Mean value of schedulers' performance 40
5.6 Experiment 3: Fairness comparison of schedulers' performance 43

Abbreviations

HDFS	**H**adoop **D**istributed **F**ile **S**ystem
LATE	**L**ongest **A**pproximate **T**ime to **E**nd
EC2	**E**lastic **C**loud **2**
GB	**G**iga **B**ytes
RAM	**R**andom **A**ccess **M**emory
TCP	**T**ransmission **C**ontrol **P**rotocol
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
SaaS	**S**oftware as a **S**ervice
IaaS	**I**nfrastructure as a **S**ervice
PaaS	**P**latform as a **S**ervice
FIFO	**F**irst **I**n **F**irst **O**ut
CPU	**C**entral **P**rocessing **U**nit
GHZ	**G**iga **H**ertz
NFS	**N**etwork **F**ile **S**ystem
ICT	**I**nformation and **C**ommunication **T**echnology
IT	**I**nformation **T**echnology
UNSTD	**U**nited **N**ations **S**cience and **T**echnology group for **D**evelopment
OFC	**O**ptical **F**iber **C**ommunication
VMs	**V**irtual **M**achine(s)
YARN	**Y**et **A**nother **R**esource **N**egotiator

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Introduction

The number of Internet users continues to grow rapidly and today's most popular applications are Internet-based applications such as, social network,video portals, e-commerce, etc. As the Internet applications serve millions of users around the globe, the amount of generated data is also huge. Users that interacts with applications generate various data such as click-stream data, crawled web documents, web requests, logs, etc. The greater the number of users that interact with system the more data is generated. For example, the number of monthly active users on Facebook; the largest social network in the world has been 1.15 billion users as of June 2013.[1] The interaction of such huge number of users with facebook application generates huge dataset. Such dataset is potentially a gold mine for the companies to understand access pattern and ad revenue of the companies[2].

Traditional database systems have difficulty to process large datasets, hence new data management and processing techniques are required to process datasets[3]. Engineers at Google were solving the same problem of building production search indexes again and again. Finally, they invented MapReduce(MapReduce was inspired by older ideas from the functional programming, distributed computing, and database communities) as a solution to build production search indexes, but it has since been used by many other industries like Facebook, Yahoo!, etc.

MapReduce is a programming paradigm used to process large datasets and Hadoop is an open source implementation of MapReduce. Hadoop runs on commodity computers

and processes the data in a distributed and parallel manner. The machines forming an Hadoop cluster runs in a master-slave pattern, where the master is called namenode and slaves are referred as datanodes. Hadoop processes the data using key/value pattern, where key is the data portion to be searched, and value is the result for searched key. The datasets are the work-units or so called "jobs" that needs to be processed by Hadoop, Hadoop splits jobs into tasks by subdividing the dataset into small, fixed-sized chunks of data, tasks are then scheduled using Hadoop scheduler for execution. The schedulers has important role for the performance of Hadoop, the literature suggested various scheduling algorithm for optimizing Hadoop, using which multiple different jobs can be served by the cluster.

Processing large dataset requires substantial computing resources, which maybe too costly for companies to buy and operate. As a solution, "Cloud Computing" is a technology that allows organization to rent the necessary computational resources and pay based on their usage. Cloud services are provided through network connection (usually through Internet), where clients can use computational resources to store, process and retrieve data according to their needs. An example of such cloud computing resources is AMAZON Elastic Cloud 2 (AMAZON EC2) cloud services, where companies can rent resources to install or run applications according to their needs. The provided resources in the cloud are virtualized environment, where more than one Virtual Machines (VMs) are placed on a single physical machine. The collocation of VMs may lead to poor performance of applications; this is because of contention of resources in the absence of perfect resource isolation. The impact of VMs collocation for Hadoop is analysed in this thesis. The new version of Hadoop (YARN) with optimized schedulers such as capacity share and fairshare schedulers, provides great opportunity for multiple organizations to submit and process different jobs. The optimized Hadoop scheduler provides the ability to divide(for example percentile based) computational resources of a single Hadoop cluster among multiple organizations, where each organization can utilize its own share of the cluster. On one hand , sharing single Hadoop cluster reduces the computational cost for the organizations, on the other hand, using a shared Hadoop cluster, the submitted jobs from one organization may result in negative impact on jobs of other organization(s).

We evaluate the performance of Hadoop scheduler based on job completion time for each scheduler and analyse the result by comparison of the results for selected set of optimized Hadoop schedulers. We also evaluate the fairness of job completion time in comparative manner for all the schedulers. The fairness is the maximum minus minimum job completion time.

1.2 Objective

Mos of the service provided by the cloud operators are not free, to use services companies may pay based on services or resources they use. Hadoop is state-of-the-art tool to process large datasets. Using optimized job schedulers of Hadoop, organizations can share use a single Hadoop cluster(also called multi-tenant Hadoop cluster) running on the cloud. As the cloud resources used by Hadoop are rented, the organizations tries to fully utilize the resources. The objective of this work is to analyse the performance of Hadoop optimized schedulers. More precisely, the performance of capacity share and fairshare schedulers of Hadoop are analysed in this thesis. For both schedulers, the effects of speculative task execution is also analysed.

For all the schedulers, the job completion time is analysed and compared. Since optimized Hadoop version provides the opportunity for multi-tenancy of Hadoop clusters for multiple organizations, the impact of collocation of datanodes is analysed. The collocation of datanodes are analysed in two cases. In the first case, two datanodes belonging to the same Hadoop cluster are collocated on the same physical machine. In the second case, two datanodes from different Hadoop clusters are placed on top of single physical machine. In both cases, performance of Hadoop is analysed to find how collocation of datanodes can affect the performance of Hadoop.

1.3 Contribution

The thesis covers the analysis of optimizations deployed in Hadoop (YARN). More precisely, this work addresses the implementation , analysis and evaluation of Hadoop performance and effects of collocation for Hadoop datanodes. Where job is the unit of work to be processed by Hadoop, the focus of the work is to analyse job completion time using different Hadoop schedulers. The time difference between various submitted jobs is another parameter analysed as fairness among job completion time. The contribution of this thesis work is:

- For a range of workloads and parameters, the collocation of datanodes has negative impact on performance of schedulers. While fairshare scheduler provides better fairness among job completion time, it has poor performance for job completion time in comparison to capacity share scheduler. For small size jobs(five GB each), the speculative task execution does not always bring better performance.

1.4 Thesis Overview

The reminder of this thesis is organized as follow. In chapter 2 the background information is explained. Chapter 3 describes Hadoop optimization and chapter 4 explains the methodology and experimental environment. Chapter 5 provides detailed information evaluation of the experiment where detail of each experiments along with results and discussions about result of experiment is explained. Chapter 6 provides analysis of cloud computing service for Afghanistan market. The last chapter, chapter 7 presents the conclusion of thesis and future work.

Chapter 2

Background

Chapter 2 explains the required background knowledge to understand the topic of the thesis. This chapter includes three section, which is about MapReduce, Hadoop and MapReduce versions.

The Authors in Google implemented many special-purpose computation paradigm in the past years. The purpose of these special-purpose computation was to process large amount of raw data such as crawled web documents, web requests, logs, etc. The process of large data helps Google to compute various graph of derived data, such as inverted indices, various graph representation of web documents, summaries of number of pages crawled per host, the set of most frequent queries in a data , etc.[4]

In a paper published in 2004, MapReduce was introduced by Google [5]. MapReduce is used to process data for applications and problems like image analysis, graph-based problems, machine learning algorithms [6].

2.1 MapReduce

MapReduce is a programming model and associated implementation to process large data sets in parallel. The approach used in MapReduce is that, for each query the entire dataset or great portion of it, is processed. MapReduce is batch query process which has the ability to run single query against whole dataset. The power is that , using MapReduce approach reasonable response time is provided for queries. Programs written in MapReduce style are automatically parallelized and executed over large set of distributed machines, which allows programmers to easily utilize resources of distributed

systems.

The MapReduce program reads input key/value pairs and generates output key and value pairs. A MapReduce program consist of map and reduce phases, where both map and reduce phases are defined as map and reduce functions written by programmers. The map function reads input data as key/value pairs and generates intermediate values. The output of map function is processed by MapReduce platform to The reduce function reads intermediate data generated from map function(s) and merges all values associated with same intermediate key.MapReduce is designed to run jobs that lasts minutes or hours on dedicated hardware in single data center, with very high bandwidth interconnects.[7]

MapReduce is some how restrictive programming model, where users are limited to key and value types. Both key and value are related in specified ways, and the mappers pass keys and values to reducers for further process.

For better understanding, an example of MapReduce logical data flow is explained in. The goal of MapReduce in below example is to find maximum global temperature recorded for each year. Weather dataset is the the data to be processed by map reduce. Map function, simply reads the data and emits the year and recorded temperature as key and value. It is good to drop bad records such as missing temperatures, suspects and erroneous in map phase. The output of map function is is not directly injected to reduce function, it is processed by MapReduce framework to sort the them by key(in this case year is the key for MapReduce). As result, each year appears with list of temperatures. Now, the reducer finds the maximum temperature for each year and store it in output file.

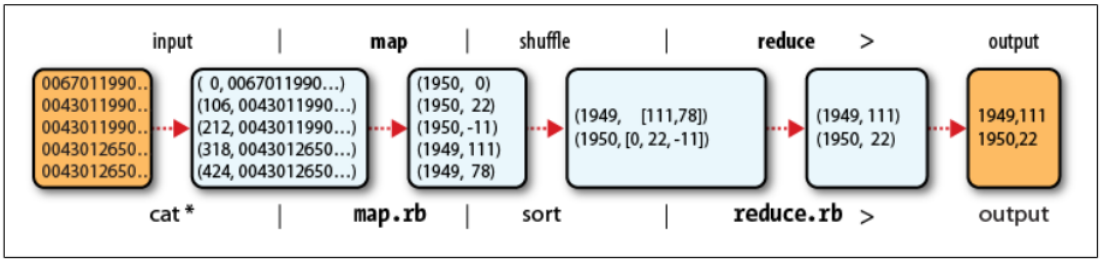

FIGURE 2.1: MapReduce logical data flow [7]

2.2 Hadoop

Hadoop is an open source implementation of MapReduce programming model used to process huge datasets. Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library[7]. Hadoop does the process of huge datasets in key/value pattern search. Where key is the data to be searched and value is the result of search for specific key.

Typically, datasets to be processed by Hadoop is large enough. Hadoop splits large data set into small fixed size chunks of data, called MapReduce jobs. To process the jobs, Hadoop splits jobs into smaller units called "Tasks", thus, a single job may consist one or many tasks. Using scheduler, tasks are scheduled for process and execution on the machines called datanodes.

Though Hadoop can run on single machine, but as it is used to process huge datasets, a set of machines forming Hadoop cluster is used to process data. The machines in Hadoop cluster works in master-worker pattern.The detail about machines in Hadoop cluster is explained in "Namenode and Datanode" section of this chapter. Hadoop Distributed File System (HDFS) is used by Hadoop to store and retrieve the data. HDFS has bigger size of data blocks then normal operating system file systems. The detail about HDFS is explained in "HDFS" section of this chapter.

There are two versions of MapReduce deployed in Hadoop, called MapReduce version 1(classic MapReduce) and MapReduce version 2(Yarn). The data process on each MapReduce version is processed using different components and techniques. The detail for each of these MapReduce versions are explained in **??**. Before explaining to detail of how does Hadoop works, here are some terms that helps the understanding of Hadoop/MapReduce data process:

Job MapReduce job is unit of work that needs to be processed by set of computers, in this case Hadoop cluster. It consist of input data (raw data to be processed) , job configuration information and MapReduce program.

Task To run the jobs, Hadoop divides it into smaller units, called tasks. Tasks are real data that is processed in the system. There are two types of tasks: map tasks and reduce tasks.

Both, map and reduce tasks are predefined user functions, that is used to process data. For example, a map task reads the input data and searches for key in that data. The result of map tasks are redirected to reducer tasks/function, to sort and write the output values.

Job Size The default job size is 64 MB, which is equal to HDFS block size. Such job size is good for Rack Locality Feature of Hadoop. The job size is configurable and depends on job type and requirement may change to bigger or smaller size. If the job size is very small , then the job creation and map creation time will dominant the over all execution of job. having many small jobs mean that the execution time for each job is smaller comparing to large input. So if we run the small jobs in parallel on Hadoop cluster, the total time of processing all small jobs will be smaller than total time to process large input data set.

2.2.1 HDFS

HDFS is the file system component of Hadoop which stores large data sets across cluster of computers in reliable and distributed manner. HDFS is designed to stream data in high bandwidth to user applications. HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware [7].According to [7] HDFS is designed to accommodate the follow:

Very Large Files The very large in this context refers to files in size of hundreds of gigabytes or terabytes.

Streaming Data The efficient data processing idea behind HDFS design was based on write-once and read many times. Typically, dataset is generated or copied from source and process/query is executed on large proportion, if not all, of dataset. Therefore the time to read the whole file is more important than reading the first record.

Commodity Computers HDFS does not require expensive highly-available and reliable hardware. It is designed to run over cluster of commodity hardware (commonly available hardware from multiple vendors) where the failure chances of hardware is very high. For case of hardware failure, HDFS is designed to continue process and work without noticing the user application from hardware failure.As the idea behind HDFS design is for large data files and the read time for the whole file is more important,thus HDFS will not work so well for low latency applications and small files.

Not for each and every data and application HDFS is fair data storage and retrieve method. Below are examples of cases where HDFS may not work well for applications that fall into below categories.

Low-latency data access Applications that requires low latency in tens of milliseconds, will not work well using HDFS. The idea behind HDFS was for data with high throughput, loading large files, where total load time of a large file is critical.

Lots of small data The information about data files stored and processed by Hadoop is stored as data called inode data. The inode data and list of blocks belonging to each file is called metadata. The namenode stores the filesystem metadata in random access memory (RAM). The number of files in namenode is limited by memory size. Each file, directory, and block takes 150 bytes in memory, so, it is feasible to have millions of files, but billions is beyond the capacity of available RAM.

2.2.1.1 HDFS Blocks

The minimum amount data or sequence of bytes (or bits) that disk can read or write is called disk block size. The size of disk block is usually 530 Bytes . To read and write data into blocks, filesystem blocks which are in size of kilo bytes are created on top of disk blocks. Generally, the disk blocks are transparent to filesystem.[7] HDFS also has the concept of block; it is default block size is 64 megabyte. The size of HDFS block can be modified to larger number for example 128 megabytes or 512 megabytes. HDFS breaks the large file into fixed size chunks called HDFS blocks. Each block is stored independently in the cluster. For small file chunks, full capacity of HDFS is not occupied by HDFS.

The time to read and write on disk depends on two factors called seek time and data transfer rate of the disk. The time needed to move HEAD (data reader or write component of disk) to the block from where it should read or write is called seek time. The amount of data that disk can read and transfer in second is called disk transfer rate which is usually calculated as megabytes per second. HDFS blocks are large compared to disk block size, this is to reduce the seek time of the disk. For large blocks, the data transfer time is significantly bigger compared to seek time to move to beginning of the block, thus time to transfer multiple files is equal to disk transfer rate.

It is not necessary that all blocks of the file to be placed on same single disk. So, an advantage of the block structure is that if a file is larger than available capacity of single disk in the cluster, it can be stored across multiple disks in the cluster.

2.2.2 Namenode and Datanode

As explained in [7], HDFS cluster consist of two types of nodes which operates in master-worker pattern: a namenode (the master)and number of datanodes (workers). The total

number of datanodes and namenode form a HDFS cluster. The namenode is responsible to store and manage filesystem namespace and the metadata for all files and directories in the tree. The information is stored in local disk in two different files "namespace image" and "edit log".

Datanodes are workers that stores and retrieves data when they are told by namenode. The datanodes updates the namenode with list of data blocks that they are storing. All nodes in a Hadoop cluster communicate with each other using TCP-based protocols(TCP-based RPC framework). To ensure reliability of data, multiple copy of data is stored on across multiple data nodes. By default three copy of the same data is stored.

All the information about filesystem is stored on namenode. The namenode knows all the datanodes on which the blocks for a given file is located. In case, if namenode data is erased (due to failure or any other reason), then, all the files on file system is lost because there is no way of how to reconstruct data blocks. There for , having resilient namenode to failure and Hadoop provides two mechanism for this.

The first way is to configure Hadoop in a way that it write the steady state of the namenode to multiple filesystems as back up copy. The writes are atomic operations that can write to local disk and to a remote Network File System (NFS) mount.

As second solution, it is possible to run another namenode as secondary namenode to merge the image-space image with edit logs in order to prevent edit log files from becoming too large. Since secondary namenode requires as much CPU and memory as namenode (primary namenode) , usually, it runs on separate physical machine.The secondary namenode keeps copy of the merged namespace image , and this copy can be used in case of namenode failure. Usually, when namenode fails, copy of namenode data which are on NFS is copied to secondary namenode and runs secondary namenode as new primary.

2.2.3 Schedulers

Hadoop uses scheduling algorithm to assign jobs/tasks to nodes and executed them. The default Hadoop scheduler used first in first out mechanism for submitted jobs. It means, the jobs were manager in queue structure, and the first submitted jobs were scheduled for execution on the cluster. After completion of first job, the second job, and so on, jobs were executed. The scheduler is one of the key factors that can improve the performance of Hadoop, thus, literature suggested optimization to Hadoop scheduler. The performance and side effects of optimized Hadoop scheduler like Capacity Scheduler,

Fairshare Scheduler and Speculative task execution is analysed in this thesis work. Detail explanation about Hadoop schedulers are covered in **??** section.

2.3 MapReduce Versions

To coordinate and manage the process and execution of jobs, few more components are used by Hadoop. The component used in Hadoop depends on Hadoop versions. Up to now, Hadoop takes advantage of two versions of MapReduce, called MapReduce version 1, and MapReduce version 2(YARN). Both versions have different components and methodology to process the jobs.

2.3.0.1 MapReduce Version 1

MapReduce Version 1, which is also called classic MapReduce has the following methodology to run jobs (brief):

- Client is the one that submits jobs to MapReduce to be processed. - The JobTracker is the component used to coordinate the jobs for process. - The TaskTracker is the component to run the tasks (as explained in **??** tasks are small splits of jobs). - A scheduler is used to schedule the tasks for process. - HDFS is used to store and retrieve the data.

The figure 2.2 explains the structure of MapReduce version 1.

Detail: Usually the data to be process by Hadoop is very large dataset. Hadoop divides this large dataset input to small fixed-size "input split or split. Each split is called MapReduce job. The splits are fed as input to user defined map function. Map functions read each record of input split and process it.

Job execution are controlled by two components of Hadoop called JobTracker and TaskTracker. JobTracker is responsible to run all jobs on system. It coordinates job execution by scheduling tasks on tasktracker. The JobTracker maintains record about status of each job and monitors the progress of each task. The TaskTracker executes tasks on nodes and sends progress report to JobTracker. In case if task execution failed, the JobTracker is the one which reschedules the task on same or different TaskTracker.

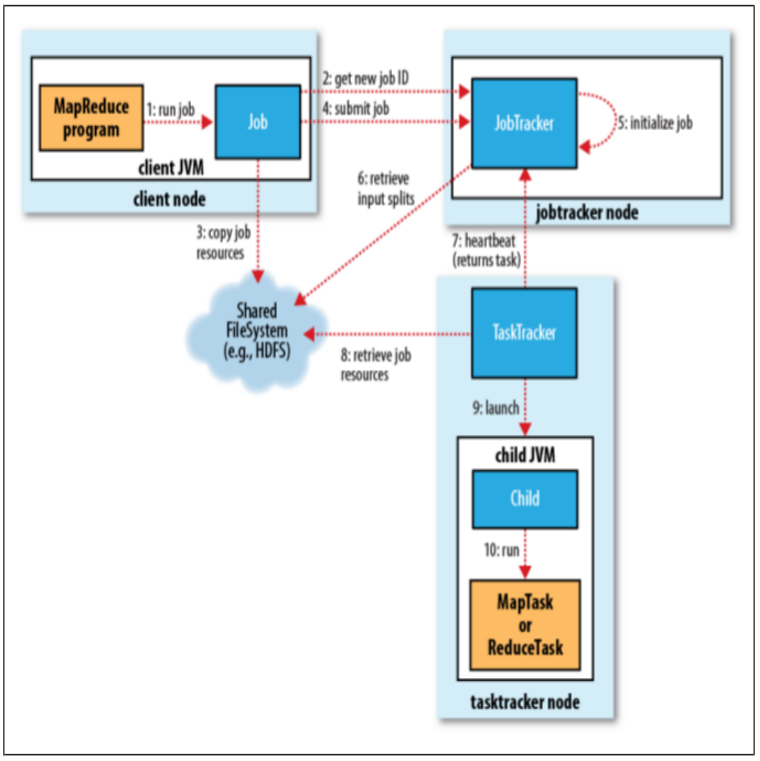

FIGURE 2.2: MapReduce Version 1 Architecture [7]

2.3.0.2 MapReduce version 2 (Yarn)

For clusters with more than 4,000 nodes, MapReduce version 1, had scalability bottlenecks. To over come this issue , the new version of MapReduce called Yet Another Resource Negotiator (YARN) was created by Yahoo! at 2010 [7].

To overcome the shortcoming of MapReduce version 1, YARN splits the functions of JobTracker into two separate components. As explained in **??**, JobTracker is responsible for job scheduling and task progress monitoring, YARN uses application manager and resource manager as two seperate daemons for these functions.

As illustrated in figure 2.3, there is single Resource Manager and per-application Applications Master. Application could be single job or group of jobs. The per-application master, negotiates for resources with resource manager and works with node manager to monitor tasks. There is an agent per datanode (worker machine) called node manager, that monitors the resource usage for application running on datanode, and submits the status report of each node to resource manager.

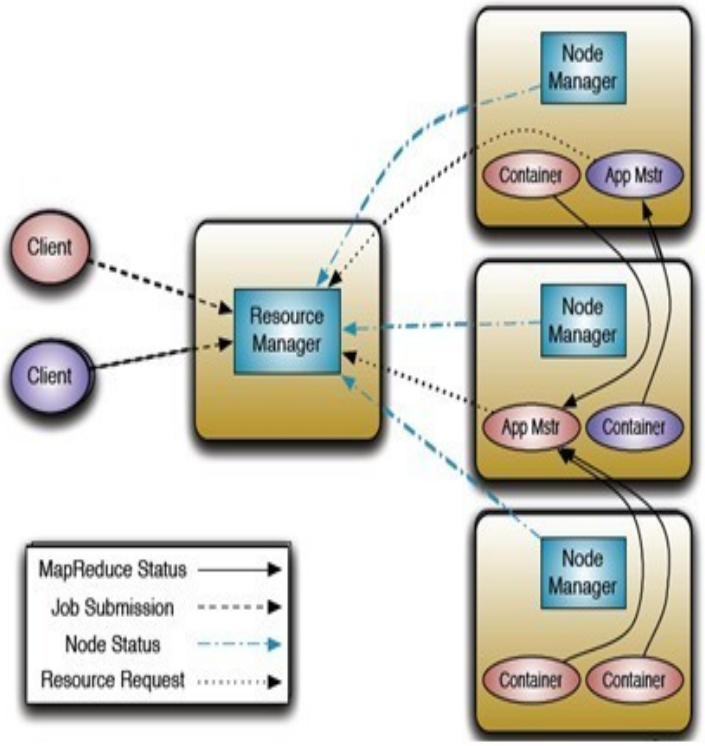

FIGURE 2.3: YARN architecture [8]

Resource manager is responsible to manage resource usage across cluster. It has two main components: The Scheduler, which is responsible to allocate resources to applications and does not perform the monitoring or tracking status of the application. The scheduler performs provides the resources based on resource requirements of applications. The abstract notion ıContainer is used for resource, where resource could be incorporate of elements like memory, cpu, disk, network etc. The scheduler has plugin policy, based on resource can be be configured and shared among different applications for example capacity and fairshare schedulers. The performance evaluation and side effects of schedulers are the core of study in this thesis.

The ApplicationManager is the entity that accepts the submitted jobs, negotiates for the container to execute per-application ApplicationMaster and in case of failure, restarts the ApplicationMaster. The ApplicationMaster is responsible to negotiate required resource container for application(job or jobs) from scheduler, track and monitor the progress of the application. The ApplicationMaster is created per application and runs for the duration of its application, and finishes after complete process of application.

YARN is compatible with MapReduce version 1, all jobs of MapReduce version just needs re compilation to be executed on YARN [8].

2.4 Cloud Computing

Cloud Computing is a distributed computing paradigm that enables the provision of available, scalable and reliable on-demand resources over a network, primarily the Internet [9]. Applications, networks, platforms , storage, processing power , service , etc can be resources in cloud computing. Resources can be provisioned and released with minimal management interaction of the provider. The composition of resources , mostly as virtualized pool of resources are provisioned to customer and can be released with minimal management interaction of provider or customer. Since, customers are unaware of how and where are the resources, the term "cloud" is an abstraction paradigm used for such services.

2.4.1 Service Models

The classification of cloud computing paradigm according to the level of abstraction and control provided to them, offers several types of services. A well known classification of cloud services , adopted by United States National Institute of Standards and Technology (NIST), is explained below:

Software as a Service (SaaS) SaaS model provides the facility to consumer to use software applications running in cloud infrastructure. To access the services, a user interface through which user can access the services from any device using applications such as web browser(e.g.,web-based email). The infrastructure on which service is running, is owned and managed either by provider or third party and consumer has limited or no control over underlying resources. Examples of SaaS providers are: Google Docs, Dropbox and GitHub.

Platform as a Service (PaaS) PaaS model allows the consumers to create,use and deploy applications on the cloud provider's infrastructure. Consumer can use programming languages and tools supported by provider to create and deploy applications. The control level given to client is more than SaaS, but limited to control over deployed applications and to configuration settings of application hosting environment. Examples of PaaS providers are: Google App Engine, Microsoft Azure.

Infrastructure as a Service (IaaS) IaaS provides the capability to the consumer to use fundamental comuting resources such as processing, storage, networks. The consumer is also able to deploy arbitrary software including operating system on the cloud. The consumer has control over software, operating systems and deployed applications,

but the physical infrastructure that runs the services is owned by provider. Typically, the physical resources are shared among multiple users or organizations by virtualization hypervisor. Examples of IaaS providers are: RackSpace and Amazon Elastic Compute Cloud (Amazon EC2).

2.4.2 Deployment Models

The classification of cloud computing can be done based on deployment models. NIST [9] defined the following four categories as deployment model for cloud computing:

Public Cloud The cloud is made available to general public or groups of people and industries. The infrastructure is either owned by cloud provider or third party.

Private Cloud The cloud is operated exclusively by single organization. The cloud maybe managed and owned by other organization or third party.

Community Cloud The Cloud is shared by several organizations that have common interests such as share mission , security or university campus. It maybe owned and managed by organization forming community or a third party.

Hybrid Cloud The composition of two or more clouds deployments by standardized technology to single cloud infrastructure , that provides portability is called hybrid cloud.

Chapter 3

Hadoop Optimizations

To improve the performance of Hadoop, literature suggested various optimizations. This chapter explains the suggested and deployed optimizations of Hadoop schedulers.

Hadoop is a MapReduce application that processes large datasets (that are divided to small fixed sizes units called jobs)on cluster of commodity computers in parallel and distributed fashion. Hadoop scheduler is the component that decides when and on which node to process the jobs, so the performance of Hadoop is closely tied to its scheduler. The optimizations like Capacity Scheduler, Fairshare Scheduler, Speculative Task Execution and Longest Approximate Time to End(LATE) are explained in this chapter.

3.1 Capacity Scheduler

Though organizations can have their own private compute resources with sufficient capacity to execute jobs on, but such private resources may be expensive and lead to low utilization of resources. Capacity Scheduler is pluggable MapReduce scheduler that is designed to allow multiple tenants to share large Hadoop cluster securely and to maximize utilization of the cluster. It is cost effective for the organization to share clusters and run jobs comparing to having their own private clusters.

Using capacity scheduler , the organizations fund the Hadoop cluster collectively, and obtains their share. The available resource in the Hadoop cluster is partitioned and guaranteed minimum share for each organization is provided. In addition, organization can access more then limited share only when the cluster resource is not used by other organization. This mechanism provides elasticity for organizations and maximizes the

cluster utilization. It means, if other organizations does not use the cluster, then any organization can use entire (not limited to its own share) cluster resources for its computational jobs as long as they are the only one organization submitting jobs to the cluster.

As the cluster is shared among organization, this leads to strong cooperation for multi-tenancy to guarantee minimum share limit of each organization. To avoid more than limit consume of resources by single job or user that affects other organization's share, capacity scheduler provides safe-guards that limits the user or job access to its share.

The capacity scheduler , manages users and jobs in queue structures. Typically the queues are setup by administrator, where multiple queues are created for job management. Each organization may own one or more queues where they can submit the jobs. Each queue is provided limited portion of computational resources to process the jobs. The amount of computational resources depends on economical share of the cluster for an organization. The more economical share you have , the more computational resources are available for your job process. Typically, the resources are divided on percentile base for each queue and if there is single organization submitting the jobs, it can use entire computational resources of cluster.

Capacity schedulers supports the following features:

Capacity Guarantee

Capacity scheduler supports multiple queues, the organizations submit jobs to their relevant queue(s). The cluster resources is allocated between all the queues based on their economical share of the cluster. It means, certain amount of resources are dedicated to each queue, where job from that queue can use these resources. There could be soft or hard limit between resources of the queues, configured by administrator. Where soft limit means that queue can access more then limited capacity if the others does not use the resource. Hard limit refers to situation where organization can use only their own share of the resources, and not more.

Security

In order to prevent unauthorized user to submit jobs to queue, strict access control lists are applied to each queue. The safe-guards can be used to ensure that users can not view or modify jobs from other users.

Elasticity

Free resources can be allocated to queues that are in demand of resources. For any queue, that has a share of computational resources, and does not need it any more or for

period of time, its share is allocated to other queues that are in demand for resources. The scheduler allocates the resources to queues that are running lower than their share and are in demand for further computational resources. This mechanism maximizes the utilization of resources and ensures that resource are available in elastic manner.

Multi-tenancy

To ensure that cluster resource is not monopolized by single, job, user or queue, limits are provided which ensures that the system (in Hadoop version 1 JobTracker) is not overwhelmed by too many tasks or jobs.

Operability

User and administrators can view current allocation of the queues of the system through console. It is also possible to change queue modification during run time without disruption to users.

Resource-based Scheduling

Resource intensive jobs are those that require or can demand for higher-requirement than default. Capacity Scheduler can accommodate applications or in particular jobs with different resource requirement. Memory is the only resource currently supported by Capacity Scheduler.

Job Priorities

Though a running job can not be preempted by any other job, but it is possible to assign higher priority to a job within the queue. Jobs that have higher priority will have access to queue's share of resources faster than jobs with lower priority. By default, priority is disabled in capacity scheduler.[**?**]

3.2 Fairshare Scheduler

Fairshare scheduler is pluggable MapReduce Hadoop scheduler that maintains separate queues for user groups (pools). Resources are allocated to jobs, in a way that on average every job gets fair share of resources over time. If there is single job running, then, it can utilize full resources of the cluster. For new submitted jobs, the slots that become free will be allocated, this mechanism provides opportunity that each job consume on average roughly the same amount of resources. Unlike default Hadoop scheduler that maintain queue of jobs, fairshare scheduler lets short jobs to complete in reasonable time and also it does not starve long jobs. [10] [11]

The fairshare scheduler maintains jobs into pools, initial fair distribution of resources across multiple pools are assigned to these pools in order to limit their access to resources. In addition to provision of fair share of resources, fair share scheduler can provide minimum guaranteed amount of resources (or computer time of CPU) to each pool to ensure that each user,pool gets sufficient amount of resources. If a pool completed its jobs and does not need the resources,excess resources is evenly distributed among other pools.[11] By default there is one queue per user so that all users can get equally same fraction of total resources.The setup of job pool is possible based on Unix user groups or any other jobconf property.Within each queue, jobs can be scheduled as first-in-first-out (FIFO) or fair share schedule. Fair-share can support job priority, where priority is weight that identifies fraction of total compute time for each job. In addition, inter-queue job priority is also supported by fair-share scheduler.[11]

Task Preemption

In the case minimum share of a pool is not provided, after waiting for certain period of time, the scheduler may kill a task from other pool(s) to provide minimum share to pool.Killing task of other jobs is called task preemption.It is also possible that preemption happen if a pool is below its half share for configurable time-out period.Usually time-out value is higher than minimum share time-out preemption.[11]

In both cases of above preemption , the fair share scheduler kills most-recently-launched tasks from over-allocated jobs, to minimize wasted computation. Since Hadoop jobs are tolerated to losing tasks, killing tasks does not cause jobs to fail but causes them to take longer to finish.[11]

3.3 Speculative Execution

The goal of speculative execution is to reduce the job completion time by speculating the tasks from straggler machines. The tasks are categorized into below 3 categories. If there is free slot on a node, then a task is selected according to the category number from one of these categories.

Failed Tasks: If a task fails multiple time , due to a bug and stop the job, such task is marked as ”failed task” and given highest priority.

Non-Running Tasks: These are fresh tasks that has not being executed on any node yet. For maps, data-locality is considered and tasks that are closer to the node is performed first.

Speculative Tasks: To find speculative task,the progress of task is monitored by Hadoop with a progress score between 0 and 1.Map progress depends on input data read and its progress score is fraction of input read data. The reduce phase compromise three sub phases where each sub-phase is counted as 1/3 of progress report. The three sub-phases of reduce phase is explained bellow:

- Fetching of map outputs, also called copy phase.
- Sorting of map outputs by key, also called sort phase.
- Applying user-defined function to the list of map outputs with each key, also called reduce phase.

In each sub-phase of reduce, the score is fraction of data processed. For example, a task halfway through the copy phase has a progress score of 1/2 * 1/3 = 1/6 , while a task halfway through the reduce phase scores 1/3 + 1/3 +(1/2 * 1/3)= 5/6.

Straggler

For map tasks and reduce tasks average of their progress score is defined as threshold by Hadoop.If progress score for a task is less then the the threshold of its category (maps or reduces) minus 0.2, and it has run for at least 2 minutes , it is marked as straggler. All the tasks below,beyond? the threshold are considered as slow and the scheduler runs at least one speculative copy of these tasks at time[2].

3.4 LATE Scheduler

LATE is the abbreviation for scheduling algorithm of Hadoop called Longest Approximate Time to End (LATE), which is robust to heterogeneity and can improve the response time of Hadoop cluster by factor of 2[2].

The idea behind LATE is to speculatively execute tasks, that are estimated to be completed in furthest time in the future. It means, the finish time for tasks are estimated and those tasks that are going to finish in longest period of time in future, are speculatively executed. Not like normal speculative execution of tasks, where speculated tasks can be run on any machine, LATE runs speculative tasks on faster nodes to complete speculative copy of task faster than original task.

To estimate the time left for task completion, the following heuristic is used by LATE: The tasks progress is monitored by scheduler, and marked with a ıProgressScore, which represents the fraction of work processed from all the work. Where map task is calculated as single task, for reduce task , the ProgressScore is the sum of ıcopy phase, sort phase, reduce phase where each phase represents 1/3 of total task progress. The ıProgressRateis another factor estimated for each task, which is equal to ıProgressScore/ T, T is the task duration time. The task completion time is estimated based on ProgressRate of each task, which is ı (1 - ProgressScore/ProgressRate) . Maybe tasks has different ProgressRate, but the assumption in LATE is the tasks has same ProgressRate.

To run speculative tasks on fast nodes, an estimation against SlowNodeThreshold is calculated for every node, any node that are below SlowNodeThreshold is considered as slow node. Speculative tasks are only executed on fast nodes (not slow nodes) . SlowNodeThreshold is the percentile of speed comparing to total speed of nodes in cluster. Choosing percentile for SlowNodeThreshold is optional, but as per [2] they observe better performance for setting SlowNodeThreshold to 25 percent.

To decide how many tasks to speculate simultaneously, a threshold called SpeculativeCap is defined. The SpeculativeCap is the total number of slots where tasks can be speculatively executed. If there is less than SpeculativeCap tasks are speculative tasks are running, and there is free node to run tasks, for task assignment the following decision is made:

- If node is slow(total progress is lower than SlowNodeThreshold) then ignore the request of node for task execution.
- Estimate the time left for running tasks that are not speculated and rank them for speculative execution.
- Run the copy of task with highest rank (lowest progress rate comparing to SlowTaskThreshold).

The decision to choose proper values for parameters of LATE has important role on overall job completion time. As per [2], they obtain best performance of LATE by setting SpeculativeCap to 10% of available task slots, SlowNodeThreshold to 25% percentile of node progress.

3.4.1 Advantages of LATE

The native Hadoop scheduler mechanism is to consider any task that is below the fixed threshold as slow task and treat them equally for speculative execution. While, LATE relaunches the slowest task and small number(at maximum as SpeculativeCap) of tasks to limit contention for shared resources.LATE mechanism is to prioritize among slow tasks on how much they hurt job response time and rank them for speculation priority. Hadoop native scheduler assumes that nodes are homogeneous and any candidate node for task execution is likely to be a fast node. In contrast, LATE takes into account node heterogeneity by ranking some nodes as slow node(nodes below SlowNodeThreshold are marked as slow node)and assigns new tasks only to fast nodes.
Hadoop native scheduler focuses on progress rate and speculatively executes any slow task. LATE focuses on estimated time left and speculatively executes only tasks that will improve job response time. For example,if task A is 5x slower than the mean but has 90 percent progress, and task B is 2x slower than the mean but is only at 10 percent progress, then task B will be chosen for speculation first, even though it is has a higher progress rate, because it hurts the response time more. Therefore , LATE provides opportunity for slow nodes to be utilized as long as this does not hurt job response time which is unlike of progress rate base scheduler that always re-executes task from slow nodes.

Chapter 4

Methodology

4.1 Objective of The Work

The focus of this work is on performance evaluation of Hadoop schedulers and side effects of collocated nodes. The goal of the work is to analyse the performance of the optimizations added to Hadoop schedulers such as capacity scheduler, fairshare scheduler including speculative task execution. Targeted set of scheduler is used as case study, and experiment and evaluations are done for specific schedulers. The study provides performance evaluation for capacity share scheduler and fairshare scheduler evaluation. In the context of this work, the performance of scheduler refers to job completion time in Hadoop cluster. The lower the job completion time is the better performance is evaluated and vice versa. Aside from job completion time as measure for schedulers performance, the fairness of job completion time is also evaluated. The analysis include findings about schedulers behaviour that causes performance degradation for job completion.

The run of virtualized Hadoop cluster, where every VM is running on one physical machine, is evaluated as baseline case. Not all the time stand alone Hadoop cluster is running on physical machines, so beside the scheduler performance in baseline, performance is analysed when two datanodes are collocated from the same cluster and when two datanodes are collocated from different clusters. The purpose of the collocated datanodes is to analyse how the placement of Hadoop nodes can affect the performance of the Hadoop.

4.2 Methodology

The goal of the work is to analyse Hadoop schedulers performance and side effects of collocated Hadoop datanodes. To achieve the goal, a placement strategy and set of schedulers are defined for the experiments. In this thesis, an experiment consist of set of schedulers, along with node placement strategy with workload execution on experimental environment.

The optimized schedulers that are selected for all the experiment are capacity share and fair-share schedulers. These schedulers are selected because they both provide the opportunity for multi-tenancy of resources among organizations. In addition to default behaviour of scheduler which performs speculative task execution, the non speculative behaviour of the schedulers are also evaluated. Overall, the scheduler set includes two schedulers in two different status which forms four cases in total:

- Capacity share scheduler with speculative task execution which is referred as "cpt" in plots.
- Capacity share scheduler without speculative task execution which is referred as "cpf" in plots.
- Fair-share scheduler with speculative task execution which is referred as "fst" in plots.
- Fair-share scheduler without speculative task execution which is referred as "fsf" in plots.

The placement strategy is about where to locate the datanodes and consist of three different cases of baseline,collocated datanodes, collocated clusters.

Baseline In this case, Hadoop nodes are placed as one node per physical machine. Not any machine is shared between two datanodes. The experiment is evaluated for the complete set of Hadoop schedulers used as baseline. The result of the experiment from baseline case is used as base performance of schedulers and two more cases are compared to baseline to find the effect of collocation of datanodes. In total there is single Hadoop cluster running on bare-bone hardware using VMs.

Collocated Datanodes This is the case where more than one datanodes are placed on one physical machine. In total two datanodes are placed on single physical machine. Both datanodes located on one physical belongs to same Hadoop cluster. The goal is to find out how collocation of data nodes from same Hadoop cluster can affect the performance of Hadoop scheduler?, In total there is single Hadoop cluster running with two datanodes on every machine using VMs.

Collocated Hadoop Clusters The collocated Hadoop cluster is similar to Collocated datanodes case with difference that it has two Hadoop clusters launched on computational resources. In this case a total of two datanodes share a single physical where each datanode belongs to a separate Hadoop cluster. The goal of this placement is to analyse the performance impact of datanode from one cluster on the datanode from another cluster. In total two there are two Hadoop clusters running with two datanodes of different cluster on every physical machine using VMs.

Fairness Fairness represents the time difference between job all job completion time. The fairness is calculated as maximum job completion time minus minimum job completion time for each run during experiment.

The total number of VMs used in baseline and collocated datanodes are equal. While in baseline every datanode VM is spawned on one physical machine and in collocated datanodes two datanode VMs are placed on one physical machine. The total number of physical machines in collocated datanodes case, used for datanodes placement is half of the number of physical machines used for datanodes in baseline case. The collocated cluster case is similar to run of two baseline at the same time. It means, each Hadoop cluster in collocated cluster case is exactly the same as single baseline Hadoop cluster.

4.3 Work Scope

The scope of the thesis is limited to few cases of Hadoop optimizations, and node placement effects. The results of the thesis is limited to experimental environment,used workloads, tools, softwares, and physical machines. The results from this thesis does not guarantee that Hadoop optimization will have same results in every other scenario or environment. Some limitation of the work is explained in next paragraphs.

This thesis addresses the performance evaluation and side effects of Hadoop optimizations. The optimization refers to optimized Hadoop schedulers like capacity scheduler, fairshare scheduler, and speculative task execution. The schedulers can be configured in different ways like queue configuration, assigning percentile of resources and so on, but this thesis work is limited to default queue configuration of the schedulers. The only change of parameter is the number of reducers changed for experiments. I select YARN as optimized Hadoop version, and tested all the schedulers using YARN.

The side effects here stands for effect of multiple nodes sharing the same physical machine. I placed two datanodes running on two VMs on top single physical machine to see how the datanodes placement have side effects on Hadoop performance. The placement of two datanodes from same cluster and from different clusters were tested to see the difference of side effects for each case.It is possible to configure namenode in a way, to act as both namenode and datanode. In the experiments , the namenode was only acting as namenode (not as datanode). The performance of scheduler is evaluated for the cases where all the datanodes are processing the data. The results of evaluation and side effects of Hadoop optimization is also limited by submitted jobs size and types.

There is single workload "terasort" used as workload and terasort benchmark is used for the performance evaluation of the schedulers. So, the results of the thesis is limited to "terasort" workload and this result may vary for Hadoop schedulers performance using other benchmarks. The thesis result is limited to results for used Ubuntu VMs on specified set of physical machine. All the experiments were executed using Ubuntu VM and it was not ran on bare-bone hardware. it is possible that running the same experiments on bare-bone hardware provide different results.

4.4 Experimental Environment

The experiments are executed on test-bed environment. The environment consist of resources: physical machines, Ubuntu operating system, Hadoop application, OpenStack and tools/scripts to process and analyse the log data.

Each experiment consist spawning and installation of Ubuntu 12.10 Virtual machine on top of physical machines. For all the experiments, Hadoop-snap-shot-3 is configured and used as instance of Hadoop application to form Hadoop cluster. After completion

of Hadoop setup, using terasort, workload is generated and stored on datanodes. Once, the workload generation is completed, the terasort starts process to sort the data using Hadoop's scheduler. The performance of each and every scheduler for every placement strategy is analysed for the time duration that terasort sorts the generated workload. A set of metrics like job start time, end time, number of mappers and reducers, number of killed tasks, etc are collected for further analysis.

For management of VMs, including installation and deletion, OpenStack software is used as tool. To automate the process of installation, configuration of VMs and nodes, and running terasort on Hadoop cluster, python scripts are used. The bash scripts collects the required metrics from the logs generated from Hadoop's performance. For further process and analysis of collected metrics, R was use as tool to analyse and plot the log data. The scripts are appended at the end of thesis to appendix.

4.4.1 Physical Resources

A total number of seven(7) computer machines connected through central switch is used to run the experiments. All the computers are connected using Gigabit Ethernet port to the switch. Each computer has sixteen(16)GB of RAM(Random Access Memory). The computers are equipped with eight(8) CPUs(Central Processing Unit), where the speed of each CPU is approximately 2,3 GHZ.The system uses 10 GB of disk space to store the virtual machine and Hadoop software. Additional mounted hard disk space of seventy two(72) GB is provided as NFS(Network File System) storage to each computer.

4.4.2 Terasort

Terasort is a benchmark tool used to sort large set of generated data. It is a standard tool to generate and sort large data sets using random data. The tool was used by Yahoo! at 2009 to sort one terabyte of data on 9000 machines using Hadoop. The two core component of terasort is Teragen and Terasort.

Teragen - Generates the random data that is used as input data for terasort.The data is generated in rows and the format of row is "¡10 bytes key¿¡10 bytes row-id¿¡78 bytes filler¿". The keys are random characters from the set .. , row-id is justified row id as a int and the filler consists of 7 runs of 10 characters from "A" to "Z". Teragen divides the number of rows by the desired number of tasks and assigns set of rows to each map.**??**

TeraSort - It is implemented as a MapReduce sort job with a custom partitioner that uses a sorted list of n-1 sampled keys that define the key range for each reduce.

4.4.3 OpenStack

OpenStack is developed by developers of cloud computing, it is open standard cloud operating system for public and private cloud operators. OpenStack consists of three core components:OpenStack Compute (code-name Nova), OpenStack Object Storage (code-name Swift), and OpenStack Image Service (code-name Glance).

OpenStack compute The OpenStack compute is designed to provision and manage large cluster of virtual machines. The software provides control panel for running instances, managing network and control of access via users.

OpenStack Object Storage The OpenStack Object Storage is used for creating petabytes of accessible data. It is a system for long term storage of large amount of static data. The data can be leveraged,updated or retrieved.For better scalability and redundancy, the data is stored in distributed manner with no central point of failure.

OpenStack Imaging Service Using image services, the clients can register new disk images.The image discovery is designed to facilitate the discovery,registration and delivery services of virtual disk to the users.

The software called "OpenStack" is used as tool to create,store and manage virtual machines. We use it for our experiments. Initially, we spawned only Ubuntu VMs and configured it for Hadoop cluster to run terasort workload. After making sure, that VMs are running properly and sorts the workload, we create an image from running VM to use it for further experiments. Basically, images are similar to clone of Ubuntu system that is capable to reboot, install, and configure for Hadoop clusters. Though, there were only Ubuntu virtual machine used for experiments but, within Ubuntu two different virtual machines images were configured which was called "Hadoop namenode image" and "Hadoop datanode image". Respectively, the images were spawned, configured and used to act as namenode and datanodes for experiments.

4.4.4 Ubuntu Virtual Machine

Chapter 5

Evaluation

This chapter presents the detail of the experiments, results, analysis and discussion of results. For better understanding of the experiments a structure is created where the experiment detail, objectives,results,fairness and discussion about every experiment is explained. The experiments are performed for three cases of baseline, collocated datanodes and collocated clusters. For every placement cases of baseline and collocated datanodes single Hadoop cluster is used to sort the generated data. The collocated cluster case was experimented using two set of six VMs (a set of six VMs per Hadoop cluster) for all schedulers. All the experiments, consists three placement cases of baseline, collocated datanodes and collocated clusters and for each placement, all of four schedulers cases "cpt","cpf","fst" and "fsf" performance is evaluated. In total, every experiment consists twelve (12) cases for all the placement and schedulers cases. Below settings were the same for all the experiments:

- Every experiment was executed five times.
- Every VM had the same amount of Hard drive, memory and CPUs.
- For each Hadoop cluster, the workload was 25 GB of data.
- Five jobs were submitted for each Hadoop cluster, each job size is 5 GB.
- Three copy of workloads were stored on datanodes.
- Total of six VMs, one as namenode and five other acting as datanodes formed a Hadoop cluster.

5.1 Experiment 1

In experiment one, we sort the generated workload using terasort, without tuning any parameter of MapReduce. In each Hadoop cluster, a total of five (5) jobs were submitted by single user, where each job size was five (5) GB of data, in total twenty five (25) GB of data. The data were replicated three (3) times across datanodes for reliability.Single user submitted five jobs to default root queue of Hadoop scheduler. There were no special configuration of Hadoop scheduler.

5.1.1 Objectives

One of the objectives of the experiment was to find out what is the default performance result for terasort workload. It means, the number of mappers and reducers are not pre-specified, and Hadoop scheduler is allowed to use arbitrary number of mappers and reducers. The experiment was designed to analyse the default behaviour of Hadoop scheduler for terasort. It is also important to see by default, which scheduler has better performance and which scheduler has better fairness in comparison to other schedulers. We also wanted to analyse how is the affect of speculative task execution (which is one of the optimizations to Hadoop scheduler) on the performance of Hadoop. The experiment also addresses the impact of collocation for default terasort workload.

As this is the first experiment set, where we did not changed any parameter, so the result of this experiment is used to see how the parameter change on the other experiments affects performance of Hadoop.

5.1.2 Results

During the experiment, we find out that by default, terasort runs with single reducer. For experiment the total number of mappers were more than forty (40+) and single reducer. It means, to complete a task , output of all mappers should wait for single reducer to become free and process the data which causes delay in overall job completion time. The figure 5.1 illustrates the mean values for job completion time for each scheduler and placement cases.

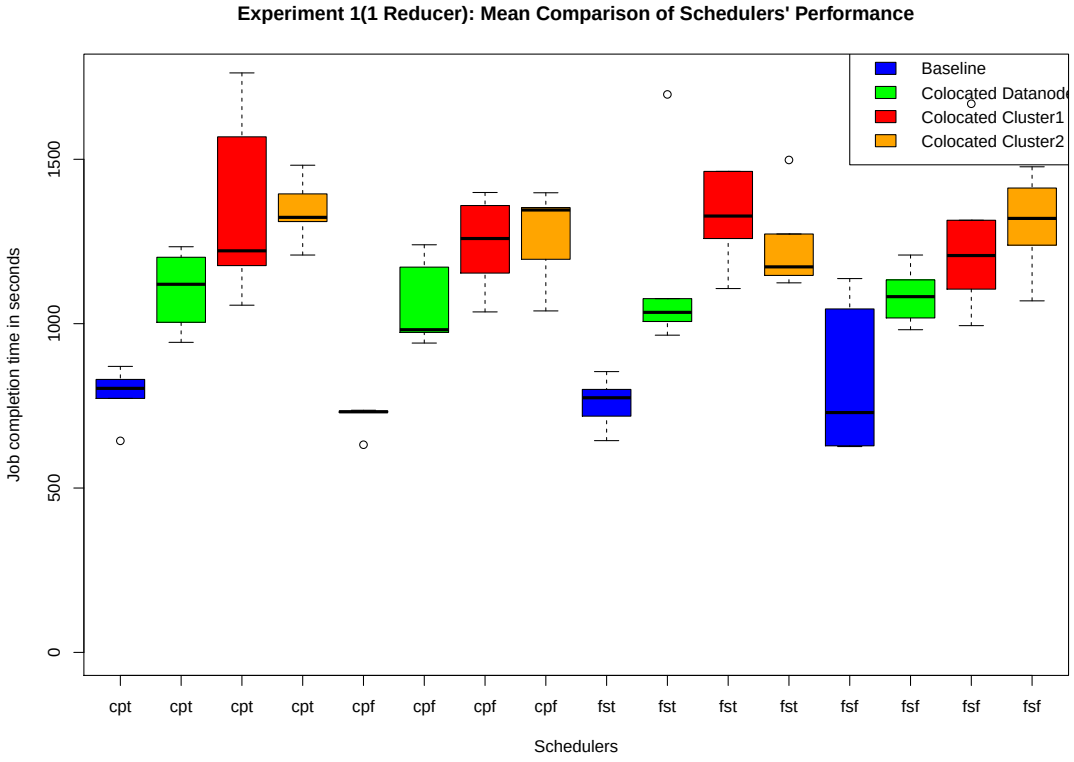

FIGURE 5.1: Experiment 1: Mean values of schedulers' performance

Capacity Scheduler with Speculative Task Execution (cpt) For "cpt", the mean values of job completion time is around 850 seconds in baseline with distribution values for job completion time (performance variance) in range of 820 to 880 seconds. The mean value increases for the case of collocated datanodes to approximately 1125 seconds on average, and performance variance interval of 1000 to 1250 seconds. The collocated cluster case has the highest values for job completion ranging from 1200 to 1600 seconds with mean value of around 1400 seconds for cluster1, and range of 1340 to 1420 seconds with average of 1380 seconds for cluster2. The time difference of "cpt" performance between baseline and collocated datanodes case is 275 seconds, while the difference between baseline and collocated clusters case is higher with value of around 550 seconds.

Capacity Scheduler without Speculative Execution (cpf) For "cpf" the mean values of scheduler performance is around 720 seconds with very small range performance variance. The collocated datanodes has the second best value for this scheduler with an average of 1125 seconds, and range of 1000 to 1250 seconds. The performance of scheduler for collocated cluster1 has an mean value of 1300 with range of 1200 to 1400 seconds, where for collocated cluster2 performance has mean value of 1310 seconds with range of 1220 to 1400 seconds. The time difference between baseline and collocated

datanodes case is 405 seconds, while the collocated clusters have higher difference of 585 seconds.

Fairshare Scheduler with Speculative Execution (fst) The "fst" scheduler has performance mean value of around 800 with range of values from 720 to 880 seconds. The collocated datanodes case has performance mean value of 1085 seconds, with range of 1050 to 1120 seconds. The performance of collocated cluster1 has mean value of 1400 seconds with range of 1320 to 1480 seconds. The performance of collocated cluster2 has mean value of around 1265 seconds with range of 1200 to 1330 seconds. The performance difference between baseline and collocated datanodes case is 285 seconds, and between baseline and collocated clusters case the difference is 630 seconds.

Fairshare Scheduler without Speculative Execution (fsf) The performance of "fsf" scheduler has mean value of around 890 and range of values from 700 to 1080 seconds. The performance of "fsf" in collocated datanodes case has mean value of 1100 seconds, with range of 1020 to 1160 seconds. The "fsf" performance for collocated cluster1 has mean value of 1240 seconds for and range of values from 1120 to 1360 seconds. The "fsf" performance in collocated cluster2 case has mean value of around 1345 seconds with range of 1260 to 1430 seconds. The performance difference between baseline and collocated datanodes case is 110 seconds, and between baseline and collocated clusters case the difference is 400 seconds.

Side-Effects of Collocation For three various node placement cases, the performance of both capacity scheduler and fairshare scheduler including the cases with and without speculative task execution are affected by collocation of datanodes. The results show that, collocation of datanodes has negative impact on performance of Hadoop schedulers and leads to longer job completion time. Though, in both collocation cases two datanodes were placed on single physical machine, but the collocated clusters case has worse results for Hadoop performance in comparison to collocated datanodes case. This is because in collocated clusters, the two nodes running on top of single machine are managed by two different namenodes, which does not know about each other, and their decisions harms the performance of collocated datanodes. It means, collocation of datanodes from different clusters does provide equal result to collocation of datanodes from the same cluster.

Speculation Effects For experiment one, the speculative task execution does not improve the performance of Hadoop schedulers. Inversely, in baseline case for capacity

scheduler the "cpf" has better performance with mean value of 720 seconds in comparison to "cpt" which has mean value of 850 seconds. For fairshare scheduler, the "fst" has better performance compare to "fsf", in baseline,collocated datanodes and collocated clusters cases. As result for experiment 1, the fairshare scheduler performance is improved using speculative task execution, while the capacity scheduler performance degraded by use of speculative task execution.

5.1.3 Fairness

The figure 5.2 illustrates the fairness of job completion time for all the schedulers. The figure 5.2 shows that "fst" scheduler has the smallest range of values for job completion time in comparison to all other schedulers in baseline. For collocated datanodes and collocated clusters cases, the results of fairness is not well distinguishable, this is because, the number of reducers used in this experiment is only one, which means, results of the mappers should wait for single reducer for further process, thus, fairness is lost for almost all the cases of schedulers and placement cases.

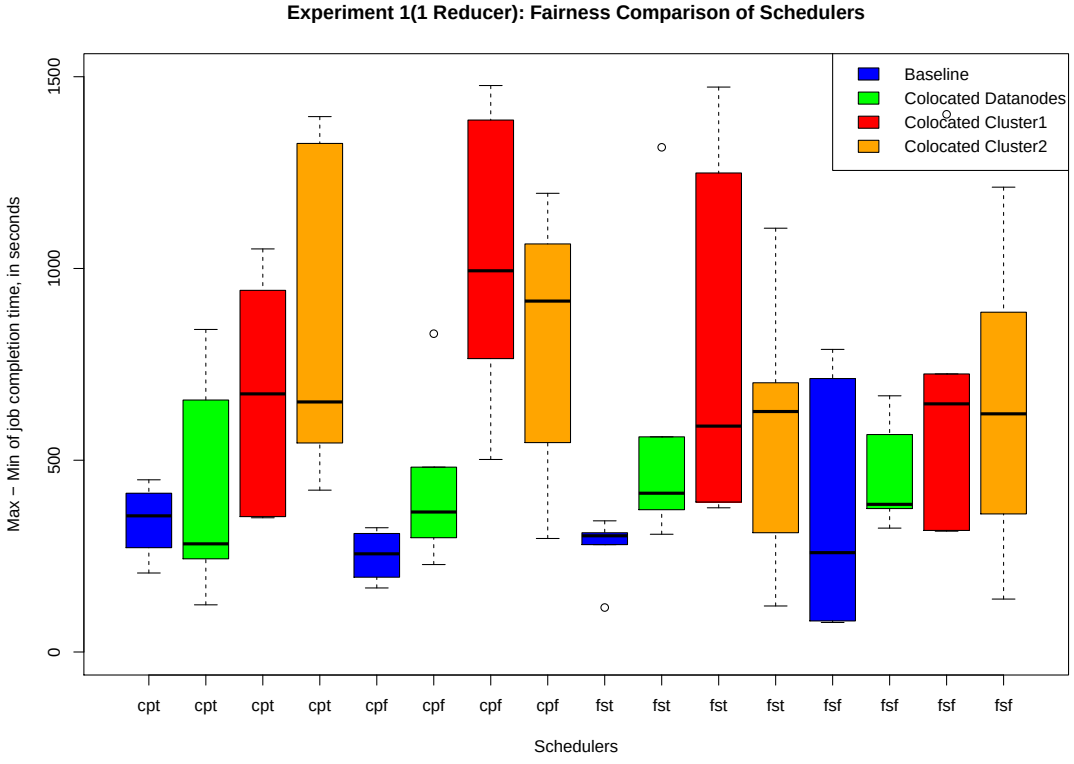

FIGURE 5.2: Experiment 1: Fairness comparison of schedulers' performance

5.1.4 Discussion

There is single reducer used by terasort to complete the sort function for all mappers. If the reducer is used to process the output of a mapper, the output of other mappers should wait till the reducer become free. Waiting for reducer to redirect the output of mapper causes noticeable delay on performance of Hadoop schedulers. Thus, the results of experiment one, does not reflect clear distinguishable different values for the performance of capacity and fairshare schedulers. The philosophy behind fairshare scheduler is to provide fairness for all the jobs submitted. The delay caused by single reducer to process output of mappers causes that, fairshare scheduler also does not have very vivid fairness in experiment one. The performance of schedulers for cases of speculative task execution and non speculative task execution are also affected almost equally by single reducer, thus, it makes it hard to judge from results of experiment one, that speculation leads to better performance of Hadoop scheduler.

The results explained for experiment one shows that, the collocation of datanodes either from same or different clusters leads to longer job completion time or performance degradation of Hadoop. Running two datanodes from the same cluster (collocated datanodes case) has better performance in compare to two datanodes from different clusters. It means, the side effects of a datanodes from different clusters is higher than side effects of datanodes from same cluster, and this is true for all the schedulers.

5.2 Experiment 2

In experiment 2, single user submits five jobs to schedulers in all sub cases of experiment. Each job consists five GB of data generated by Teragen (total of 25 GB data) and need to be sorted by terasort using Hadoop. Three copy of data were replicated and stored using HDFS across datanodes of Hadoop cluster. There is no queue configuration for Hadoop schedulers and default root queue is used for submission of all jobs. We changed (in contrast to experiment 1) the settings of experiment two and configured 10 reducers to participate in data processing.

5.2.1 Objectives

As explained in discussion part of experiment one, single reducer causes considerable delay in job completion time for all Hadoop schedulers. The objective of experiment 2 is

to see how the number of reducers has affect on job completion time or performance of Hadoop. In experiment two, by increasing the number of reducers to ten, our expectation was to have better performance evaluation in comparison to experiment one. It is also important, to see how the performance of Hadoop is improved by increased number of reducers for collocated datanodes and collocated clusters.

5.2.2 Results

The figure 5.3 illustrates the mean value results for all the experiment cases. Over all , the job completion time is better than experiment one, it means number of reducers have important role in Hadoop performance.

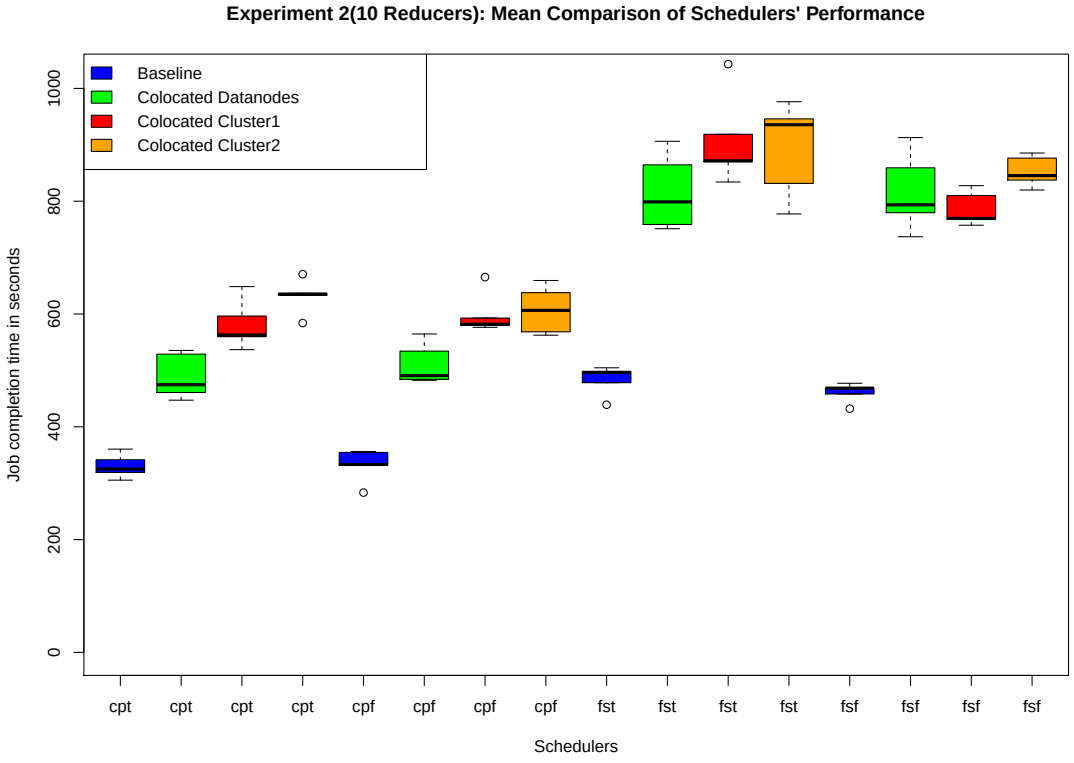

FIGURE 5.3: Mean value of schedulers' performance

Capacity Scheduler with Speculative Execution (cpt) The mean value for "cpt" performance in baseline case is approximately 300 seconds, which is the best value among all schedulers cases. For collocated datanodes, the scheduler performance has mean value of approximately 500 seconds, which shows that collocation of datanodes increased the job completion time 70% more than the baseline. For collocated clusters cases, the

scheduler performance has mean value of approximately 600 seconds (approximate average for both clusters), which is double value of job completion time in comparison to baseline and shows 100% increment job completion time.

Capacity Scheduler without Speculative Execution (cpf) The mean value for "cpf" performance in baseline case is approximately 330 seconds, which is worse than "cpt" (which was 300 seconds), but the second best result among all the schedulers. The side effects for collocated datanodes case of "cpf" is almost similar to "cpt". The collocation of datanodes has an average performance of approximately 530 seconds for job completion time, which shows 70% increment in job completion time in comparison to baseline. The average performance for collocated clusters is approximately 600 seconds for both clusters, which shows 95% increment in job completion time.

Fairshare Scheduler with Speculative Execution (fst) The mean value for "fst" performance in baseline case is approximately 500 seconds, which is the worst mean value among all schedulers' performance. The "fst" performance in collocated datanodes case has mean value of around 800 seconds, which shows 60% increment and time difference of 300 seconds for job completion time in comparison to baseline. The "fst" performance in collocation of datanodes case from different Hadoop clusters has mean value of around 900 seconds, which shows 80% increment (400 seconds time difference of job completion time) in comparison to baseline case.

Fairshare Scheduler without Speculative Execution (fsf) The mean value for "fsf" performance in baseline case is approximately 480 seconds, which is better than "fst" and third best value among all the schedulers. The "fsf" performance in collocation of datanodes case has mean value of approximately 800 seconds for job completion time, which shows 66% increment and 320 seconds time different in comparison to baseline case. The "fsf" performance in collocated datanodes case from different clusters, has mean value of around 800 seconds (approximate value for both clusters), which is the same result as collocated datanodes case in comparison to baseline.

Side-Effects of Collocations The results in figure 5.3 show, that fairshare scheduler has worse performance for the collocation cases in comparison to capacity share scheduler. The mean value difference between baseline and collocated datanodes for "fst", which shows the side effects of collocation, is around 300 seconds, while this value is around 200 seconds for "cpt" case. Also, the mean value difference between baseline and collocated clusters cases, for "fst" is 400 seconds, while this value is 300 seconds in

"cpt" case.

For cases, where tasks are executed by schedulers without speculation (like "cpf" and "fsf" cases), still , capacity share scheduler has better performance for collocation cases in comparison to fairshare scheduler. The mean value difference for between baseline and collocated datanodes for "cpf" is 200 seconds, which is better than the mean value difference of 320 seconds for "fsf" case. The mean value difference between baseline and collocated clusters cases for "cpf" is 270 seconds, for which "fsf" has higher difference of 320 seconds.

Speculation Effects By having a deeper look to mean values in figure 5.3, it is evident that speculative task execution improved the performance of capacity share scheduler (see the "cpt" case), while decreased the performance of fairshare scheduler. Though the performance improvement of speculative task execution is minor, but still the improvement for "cpt" is consistent and true for the collocated cases in comparison to "cpf" case. In contrast to capacity share scheduler, the fairshare scheduler with speculative task execution, in "fst" case has worse result in comparison to "fsf" case.

5.2.3 Fairness

The figure 5.4 illustrates the fairness for job completion time, among submitted jobs for each placement and scheduler cases. The results show that overall fairshare scheduler has much better fairness in comparison to capacity share scheduler. The fairness for job completion time ranges from around 50 to 200 seconds for all the placement cases of fairshare scheduler, while the results for capacity share scheduler ranges from 200 to 400 seconds.

For "cpt" in baseline case, the fairness values for job completion time is in the interval of 210 to 260 seconds, which shows at least 210 seconds difference. While for both "cpt" and "cpf" in baseline case, the fairness values is lower than 300 seconds on average, the collocation had negative impact and increases the fairness values to higher than 300 seconds, and even close to 400 seconds for collocated datanodes case of "cpt".

5.2.4 Discussion

By increasing the number of reducers from one (in experiment one) to ten (in experiment two), we see that the performance of Hadoop schedulers are improved for all the

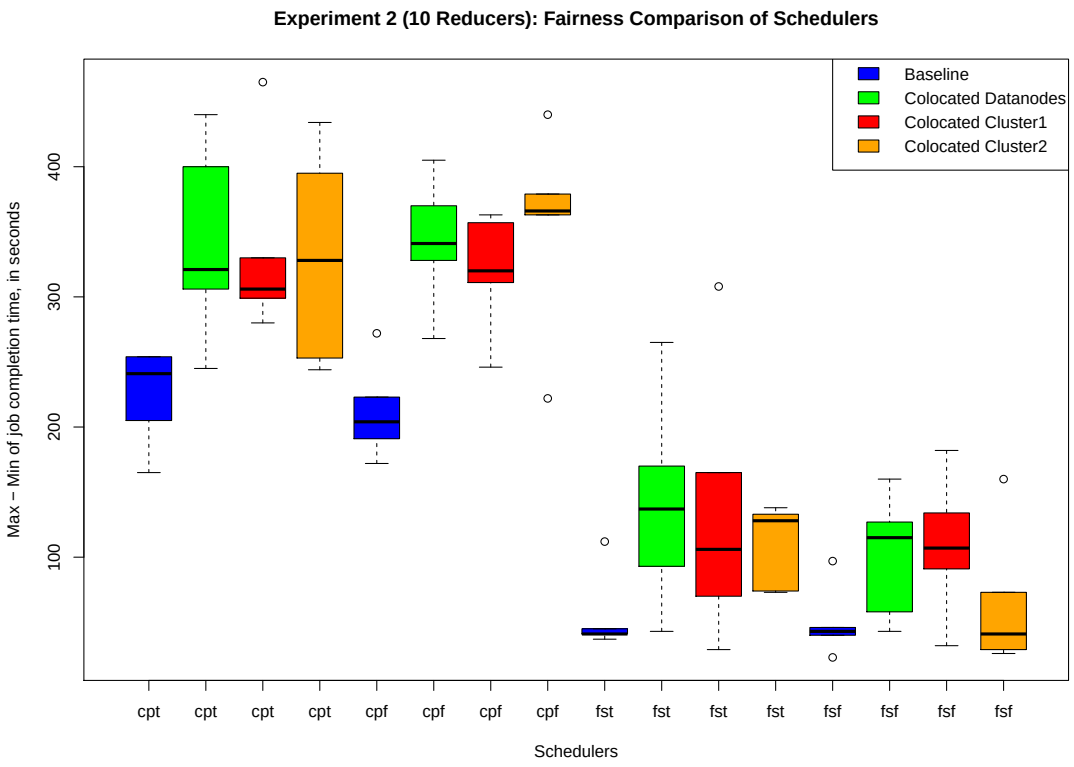

FIGURE 5.4: Experiment 2: Fairness comparison of schedulers' performance

cases. Which means, for process of workload using Hadoop, the selection of fair number of reducers plays a key role on performance of Hadoop. The number of mappers were consistent for all the schedulers, it was 40 mappers. In case of speculation of schedulers, some tasks were speculatively executed, which added speculative number of mappers and reducers.

The results shown for experiment two, indicates that capacity share scheduler has better performance in comparison to fairshare scheduler, and executes the submitted jobs in smaller time interval than fairshare schedulers. The performance of capacity share schedulers is better than fairshare schedulers in both cases of speculative and non speculative task execution, and all collocation cases.

The collocation cases have impact on schedulers performance and degrades the performance of Hadoop schedulers. The side effects of collocation for fairshare schedulers are higher than capacity share. The side effects of collocated datanodes is not as much as collocated clusters. This is because, in collocated datanodes, both nodes are controlled by single namenode,which mean, namenode is aware of the status of namenode using

resources manager. In collocated clusters cases, the collocated datanodes are belong to separate namenodes that are not aware of status of collocated datanode, so the performance of datanode from cluster one effects the performance of datanode of cluster two.

The fairshare schedulers provide better fairness performance for submitted jobs. Considering the performance evaluation of and fairness evaluation of schedulers, the fairness has cost for fairshare schedulers and that causes performance degradation of Hadoop. One may use capacity share to have better performance and fairshare scheduler to guarantee better fairness for job process.

5.3 Experiment 3

The results from experiment two show that increasing the number of reducers improves the performance of Hadoop. In experiment three, we increased the number of reducer to twenty (20) reducers, which is 50% of the number of used mappers. Rest of the experiment settings were the same as experiment one and two, where single users submitted five jobs, each job consist of five GB of data. There were no special queue configuration for Hadoop schedulers. The experiment was executed for all the placement cases and schedulers cases.

5.3.1 Objective

The objective of the experiment was to find out the effects of increased number of reducers on performance of Hadoop schedulers. The side-effects of collocation and evaluation of performance improvement for the schedulers were the other objectives. The improvement of fairness for schedulers and performance improvement of schedulers by speculative task execution were the other expected outcome of the experiment.

5.3.2 Results

By increasing the number of reducers to 20 reducers, the evaluation shows that performance of Hadoop is increased across all the schedulers. While the effect of changing the number of reducer for experiment to experiment two was very vivid and clear, the effect of changing the number of reducers from ten to twenty in experiment three is not so high in comparison to experiment two. This means, that having so many reducers does not improve the performance of Hadoop and we may select a fair number schedulers (25%

to 50% of mappers limited to our experiment environment and settings) for Hadoop. The figure 5.5 illustrates the mean values of job completion time for schedulers in all the placement cases.

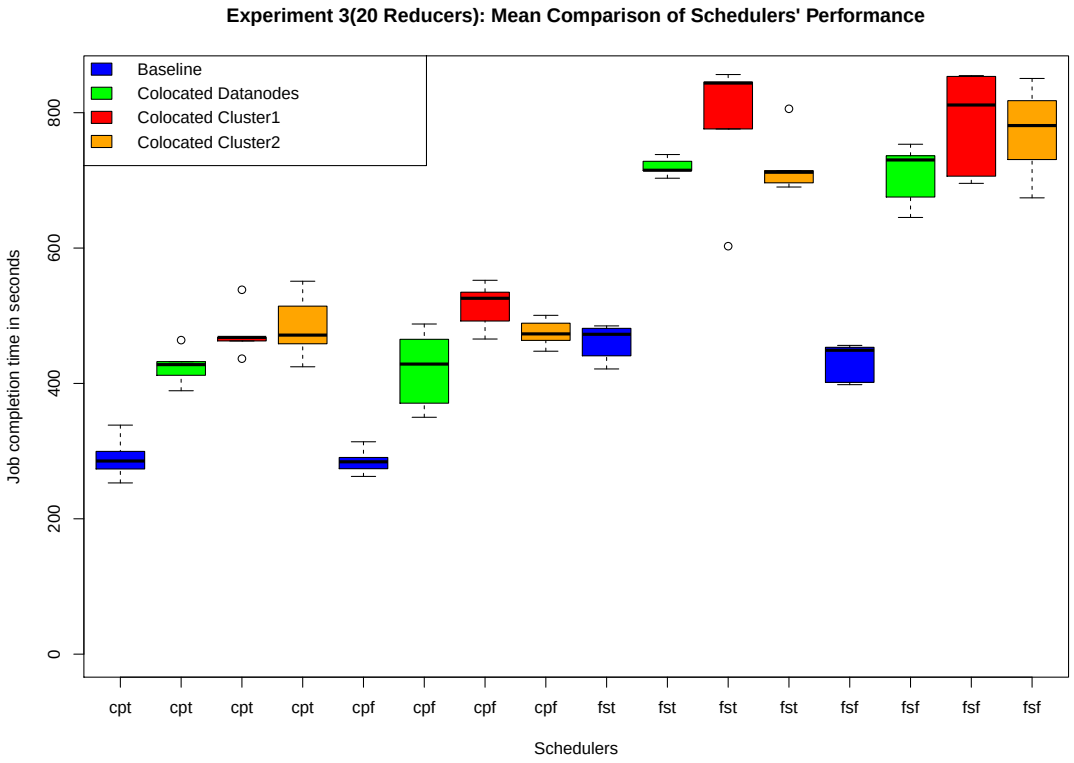

FIGURE 5.5: Experiment 3: Mean value of schedulers' performance

Capacity Scheduler with Speculative Execution (cpt) The mean value for "cpt" performance in baseline case is approximately 290 seconds. For collocated datanodes, the scheduler has mean value of approximately 420 seconds for job completion time, which shows that collocation of datanodes increased the job completion time 45% more than the baseline. For collocated clusters case, the scheduler has job completion time of approximately 500 seconds (approximate average for both clusters), which has time difference of 210 seconds in comparison to baseline and shows 72% increment in job completion time.

Capacity Scheduler without Speculative Execution (cpf) The mean value for "cpf" performance in baseline case is approximately 290 seconds, which is same as "cpt", but the distribution range of job completion time is smaller in comparison to "cpt" . The effects of datanodes collocation on median job completion time in the "cpf" case is similar to "cpt", but the job completion time is scattered in larger interval of time. The

collocation of datanodes has an average of approximately 430 seconds for job completion time, which shows 48% increment in job completion time in comparison to baseline. The average job completion time for collocated clusters is approximately 520 seconds for both clusters, which shows 79% increment in job completion time.

Fairshare Scheduler with Speculative Execution (fst) The mean value for "fst" performance in baseline case is approximately 470 seconds, which is the worst mean value among all schedulers' performance in baseline. The collocated datanodes, has performance with mean value of around 710 seconds, which shows around 51% increment and time difference of 240 seconds for job completion time in comparison to baseline. The collocation of datanodes from different Hadoop clusters are very different, cluster1 has a mean value of around 800 seconds, and cluster2 has mean value of 700 seconds. The mean value for both clusters is around 750 seconds, which shows around 60% increment with 280 seconds time difference of job completion time in comparison to baseline.

Fairshare Scheduler without Speculative Execution (fsf) The mean value for "fsf" performance in baseline case is approximately 430 seconds, which is better than "fst" and third best value among all schedulers. The collocation of datanodes, has mean value of approximately 700 seconds for job completion time, which shows around 63% increment and 270 seconds time different in comparison to baseline. The collocated datanodes from different clusters have mean value of around 750 seconds (approximate value for both clusters), which shows around 74% increment with time difference of 320 seconds for job completion time in comparison to baseline.

Node Placement Effects Similar to experiment two, the side effects of collocation for both collocated datanodes and collocated cluster cases is high for fairshare scheduler in comparison to capacity share scheduler. For capacity schedulers the mean value of baseline in comparison to collocated datanodes and collocated clusters cases show the respectively time difference values of 135 and 220 seconds. While, for similar comparison the fairshare schedulers have time difference values of 255 and 300 seconds.

Across all the schedulers, the mean values for collocated datanodes is lower than mean value for collocated clusters. This proofs the fact, that collocation of datanodes from same cluster has less side effects on collocated datanode in comparison to collocation of datanodes from different schedulers.

Speculation Effects The illustrated results in figure 5.5 shows that the speculative task execution does not improve the performance of schedulers in all the cases. For "fst"

in baseline case, the speculative task execution leads to worst result in comparison to "fsf". The comparison of mean values for "cpt" and "cpf" shows that there are no vivid performance improvement between speculative and non speculative task execution in all the placement cases.

5.3.3 Fairness

The figure 5.6 illustrates the fairness for job completion time, among submitted jobs for each placement and scheduler cases. As the results show, the fairshare schedulers have values lower than 250 seconds, in contrast , the fairness values for capacity share schedulers are higher than 250 seconds. The "fsf" scheduler has the best fairness value of 50 seconds, with small range of of distribution for job completion time. The fairness values shown for fairshare schedulers in collocated cases indicates that, collocation of datanodes has negative impact on fairness of schedulers, and leads to larger range for job completion time.

The results shown for capacity share schedulers in all the placement cases explains, that they have higher value of 250 to 500 seconds. Both collocation cases (collocated datanodes and collocated clusters) have almost similar fairness values with mean of around 450 seconds for almost both "cpt" and "cpf", while "cpf" has larger range values for job completion time.

5.3.4 Discussion

Though increasing the number of reducer lead to better performance in experiment two, in experiment three by increasing the number of reducers to twenty, which is 50% of mappers (total of 40 mappers were used), the performance improvement is not so much. It means, selection of high number of reducers does not improve the performance of Hadoop, and we must select a fair number of reducers. The results from all three experiments (experiment one, two and three) for which the number of reducers were respectively 2.5%, 25% and 50% of mappers, the experiment three had best results for all the schedulers in all the cases.

In regard to performance evaluation of schedulers, in one hand, the capacity share scheduler has the best results for both "cpt" and "cpf" cases. on the other hand, the capacity scheduler has worst fairness results in comparison to fairshare scheduler. Inversely, the

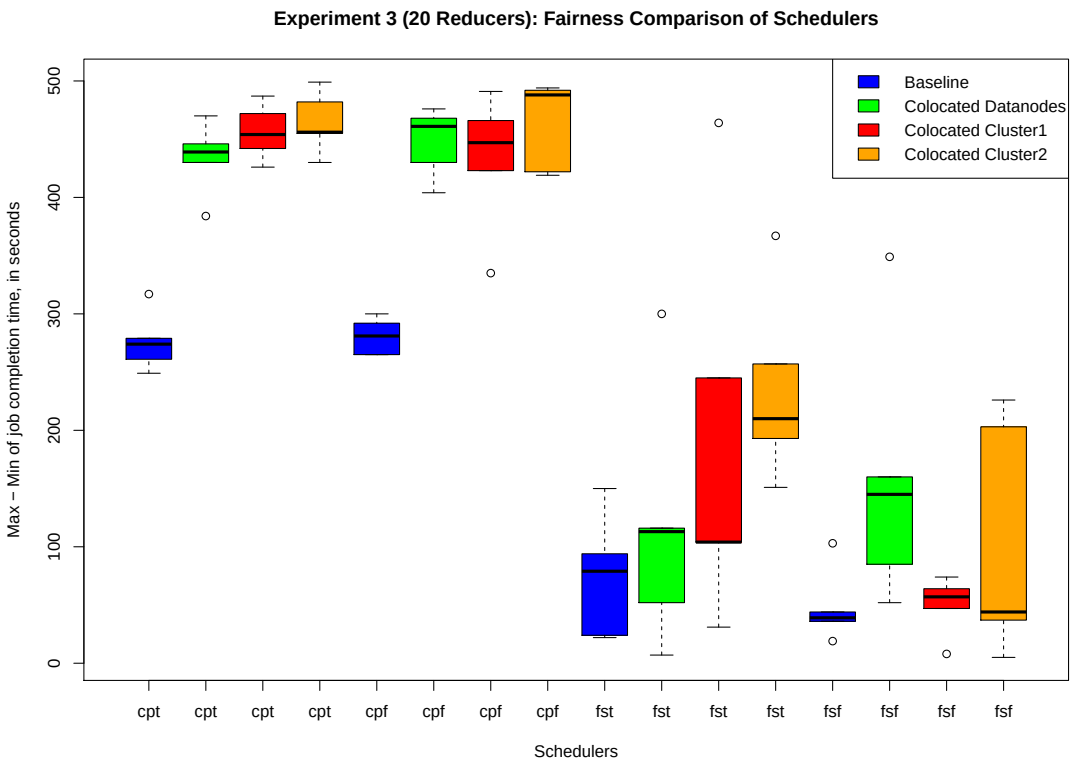

FIGURE 5.6: Experiment 3: Fairness comparison of schedulers' performance

fairshare scheduler has the worst performance for job completion and best fairness values in comparison to capacity share scheduler.

Similar to two previous experiments, the collocation cases have negative impact on performance of schedulers. In comparison to fairshare scheduler, the capacity share scheduler has more tolerance to collocation of datanodes in both cases of collocated datanodes and collocated clusters and has better performance. The capacity share scheduler also have minor better performance while speculatively executing the tasks, while in fairshare scheduler the speculative task execution causes minor degradation to the performance of scheduler.

Chapter 6

Cloud Computing in Afghanistan

Afghanistan is land lock country which most parts are enclosed by mountains. Large number of Afghan people scattered in and out of Afghanistan during last 3 decades of war. This made the Information Technology and Communication(ICT) services and facilities an essential issue for Afghanistans immediate reconstruction. After the Temporary Government in 2001 established in Afghanistan, the country obtained new view of political and socio-economic rehabilitation and structure.

6.1 Afghanistan and ICT

In the last 2 decades (80s and 90s), socio-political upheavals and war not only destroyed Afghanistans infrastructures and wealth, but it also destroyed the ICT system. By establishment of the new government after 2011, Afghan government started to encourage the private sector to do investment in Afghanistan's ICT projects. Parallel to the investment of private sectors on ICT field the government also started to develop ICT infrastructure and services in Afghanistan. As result of encourages big investors like Afghan Wireless Company (AWCC), Roshan, Areeba and Etisalat entered to ICT market of Afghanistan. In order to develop ICT and to fulfil the human resource requirements of the ICT field in Afghanistan the Afghan government Established Computer Science Faculties in many universities of Afghanistan, ICTI institute , creation of ICT Law and ICT strategy plan. Almost 80 percent of Afghan people have access to telecommunication that indicates rapid growth of ICT and implementation of ICT projects. In this section our focus is more on national projects which are offered or going to be offered by Afghan government.[12]

Afghan people need to strengthen and diversify their economies, educate and engage their young people, develop the infrastructures that support economic growth, and lure back the educated professionals and business-people who have fled to other countries. ICT will be instrumental in meeting these challenges, but recent history shows that Afghanistan is suspicious of, and resistant to, technological change. Based on a report by the United Nations Science and Technology Group for Development (UNSTD), ICT strategies are often developed and publicized mainly to attract external investment to construct new infrastructures or to market hardware and software without giving sufficient attention to local concerns and requirements [13].

6.1.1 National Optical Fiber Backbone

This project will install a national backbone network across the country, which will support all the other projects (digital lines, microwave, etc). It will allow a high volume of national and international traffic and will connect major provinces and to neighbouring countries. This Network will link many of the principle cities of Afghanistan following the route of the national roadway system. This project is for the turnkey construction and operation of the complete Optical Fiber Communication (OFC) ring around Afghanistan which is estimated approximately 3.200 km length. In addition to linking many of the key cities. This project also calls for the construction of accesses to the backbone for the other cities and provinces not directly on the backbone route. Those access routes for the other cities and provinces will be taken up separately. There will be other major spurs off this main loop connecting to neighbouring countries Iran, Pakistan, Uzbekistan, Tajikistan, and Turkmenistan.[12]

6.2 Challenges

Although, Afghan ICT had good progress in past years, but still there are challenges that needs time to be solved. Low levels of education and literacy, poor technology infrastructures,lack of ICT skilled human resources, lack of political interest to ICT and a wide gap between the disposable income of the relatively few have more and more have less income can be counted as number of challenges . Security is a big challenge that not only ICT project implementation but over all Afghan government suffer from. Use of the Internet requires a fairly complex set of skills and technology. At very least, one must have electricity, a communications line, a terminal capable of interacting across the communications lines, and (in most cases) a reasonable fluency in English [14]. Cultural issues may be important in determining the use of the ICT in Afghanistan.

6.3 Cloud Computing In Afghanistan

For the organizations invested in ICT market of Afghanistan, poor technology infrastructure, lack of security , required energy resources, IT experts all these challenges makes it difficult to establish and maintain large data centres of their own for storage and process of their data.

The poor infrastructure of ICT causes unstable Internet connections, which makes it difficult to even go for cloud services in Afghanistan. After completion of national projects in Afghanistan, such OFC project, the organizations may have stable Internet connection that will ease the use and cultivation of cloud services among organization. The organizations may rely on cloud computing services where all the infrastructure, resources, storage, process and maintenance is provided by cloud operator to the clients via an Internet connection. The organization only need Internet connection with high enough speed to access the cloud services/infrastructure and use it according to their need. Using cloud services, the organizations are free of headache for maintenance, physical resources, security and hiring experts to manage IT resources.

Chapter 7

Conclusions

7.1 Conclusions

The experimental results explained in chapter 5 gives the idea that Hadoop optimizations does not improve the performance of Hadoop for every dataset process. For cloud services where physical computational resources are shared among multiple organizations to run Hadoop cluster(s), using capacity share scheduler of Hadoop is better choice in comparison to fairshare scheduler. Overall, the capacity scheduler of Hadoop has good performance in term of job completion time, but it has poor fairness results for job completion time.

As the name explains, fairshare scheduler of Hadoop has better fairness in all the cases in comparison to capacity share scheduler of Hadoop, but it has poor performance than capacity share scheduler. For submitted jobs to Hadoop cluster from single or multiple organizations, in case fairness is important, than fairshare scheduler is better choice, otherwise , capacity scheduler has better performance.

The collocation of more than one VM acting as datanode on a single physical machine, degrades the performance of Hadoop. Collocation of VMs (datanodes) from same cluster has better results in comparison to placement of VMs from different Hadoop clusters. Though the performance of Hadoop schedulers are degraded by collocation, but capacity share scheduler is more robust to collocation of datanodes.

The thesis explored the analysis of speculative task executions by Hadoop schedulers. While the expectation is to increase the performance of Hadoop by speculative tasks

execution, the evaluated results in chapter 5 shows that this assumption is not hold always. This is because, speculative execution of tasks requires more resources to be utilized, utilization of resources decreases the performance of Hadoop for small jobs.

The selection of fair number of reducers plays important role on performance of Hadoop. The results evaluated in chapter 5 explains the idea that selection of number of reducer from 25% to 50% of mappers number leads to best result for the mapper, reducer combination.

Considering the current situation of Afghanistan, where ICT infrastructure is very poor, security and lack of experts for the IT fields are challenges, running the services on cloud operator has difficulty of unstable Internet. The projects such as NFO brings the hope to have stable infrastructure for the ICT where organization can rely on cloud services, but there are number of security and political challenges to the completion of the ICT projects.

7.2 Future Work

We addressed the evaluation and analysis of of Hadoop performance for set of schedulers using Terasort workload. We tune number of reducers, to see the impact of change on performance of Hadoop. There are possible extension to this analysis, where one can run similar experiments with different workloads and jobs sizes, to validate the performance evaluation for all the cases. Also, it is possible to tune any other parameter of Hadoop such as queue configuration of schedulers, submit of various job sizes, resource partitioning, etc, and analyze the performance of results for optimized Hadoop schedulers. The effect of collocation can be analysed for the cases where Hadoop nodes are collocated with other application in the data-centre. The test of similar experiment on bare-bone physical machines are another research field, because All the experiments are executed on VMs, which possibly may not provide the same results for execution on top bare-metal physical machines.

Bibliography

[1] Facbook. Key facts, September 2013. URL https://newsroom.fb.com/Key-Facts.

[2] Anthony D. Joseph Randy Katz Matei Zaharia, Andy Konwinski. Improving mapreduce performance in heterogeneous environments. *USENIX Association, 8th USENIX Symposium on Operating Systems Design and Implementation*, I:13, 2008.

[3] Yin Huai Zheng Shao Namit Jain Xiaodong Zhang Zhiwei Xu Yongqiang He, Rubao Lee. Rcfile: A fast and space-efcient data placement structure in mapreduce-based warehouse systems. *IEEE*, I:1199–1208, 2011.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Google, Inc*, I(2):1236–1239, March 2004.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Google*, I:all, Dec, 2004. URL http://labs.google.com/papers/mapreduce.html.

[6] Apache. Apache mahout. *Apache Mahout is a project to build machine-learning libraries (such as classification and clustering algorithms) that run on Hadoop*. URL http://mahout.apache.org/.

[7] Tom White. *Hadoop:The Definitive Guide*, volume Third Edition. May 2012. URL http://link.aip.org/link/?RSI/72/4477/1.

[8] The Apache Software Foundation. Apache hadoop nextgen mapreduce (yarn). website, August 2013. URL www.apache.org.

[9] Peter Mell Timothy Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, Draft:7, January 2011.

[10] Thomas Sandholm and Kevin Lai. Dynamic proportional share scheduling in hadoop. *Social Computing Lab, Hewlett-Packard Labs, Palo Alto, CA 94304, USA*, I:110–131, 2010.

[11] X. Hadoop fair scheduler design document. *X*, I:1–11, 2010.

[12] Ministry of Information and Communication of Afghanistan. publication and projects. Website, January 2012. URL http://mcit.gov.af/en.

[13] Elham Ghashghai and Rosalind Lewis. Issues affecting internet use in afghanistan and developing countries in the middle east. http://www.rand.org/publications/IP/IP23.:all, march 2003. URL http://www.rand.org/publications/IP/IP23.

[14] Internet World Stats. Asia internet usage and population, Dec 2011. URL http://www.internetworldstats.com/stats3.htm#asia.