

# BETTER SELECTION OF VIRTUAL MACHINES FOR A MAPREDUCE ENVIRONMENT

---

A Thesis  
Submitted to  
the Temple University Graduate Board

---

in Partial Fulfillment  
of the Requirements for the Degree of  
MASTERS OF SCIENCE

---

by  
Adam Pasqua Blaisse  
July, 2015

---

Examining Committee Members:

Jie Wu, Thesis Advisor, Dept. of Computer and Information Sciences  
Chiu C. Tan, Dept. of Computer and Information Sciences  
Justin Y. Shi, Dept. of Computer and Information Sciences

©  
Copyright  
2015

by

Adam Pasqua Blaisse

All Rights Reserved

# ABSTRACT

Better Selection of Virtual Machines for a MapReduce Environment

by

Adam Pasqua Blaisse

With the increase in the availability of large sets of data, comes the need for better and more sophisticated methods of handling and processing these sets. Due to the size and complexity of these data sets, many users have moved to using distributed systems for storage and processing. With a distributed system, there are many different things that become much more complex and many more opportunities present themselves for issues. Out of this rose the paradigm of MapReduce.

The basic idea of MapReduce is to minimize the work of the programmer and remove a lot of the chances of creating an error because of the distributed computation. To do this all the work is either done in the Map Phase by the Map Tasks or in the Reduce Phase in the Reduce tasks. Communication and synchronization is taken care of by Map Reduce so that users are protected from misusing them.

Users may also want to use map reduce along with cloud computing. The most common resource that is rented from Amazon EC2 is virtual machines. Amazon offers many different sizes with different types of configuration. Some machines may be more specialized to handle CPU based jobs, while others might be optimized for memory or disk based jobs. Each of these different VM's comes with varying levels of CPU cores, RAM, and Storage capacity to match it's use. Each of these virtual machines also has its own cost per hour.

This means that simply selecting the largest or strongest machine may not be the best option if one is trying to get the most for their money. The other resource that amazon offers is called elastic storage blocks. The basic idea of the the elastic storage blocks, is to offer a users more storage capacity to add to their virtual machine.

Like the virtual machines, storage space is has a per hour cost that depends on the amount of space used, as well as type of storage requested. These storage volumes, once purchased, can be attached to a virtual machine and used as extended storage capacity.

For this thesis, we will look at how users can best select they type of virtual machine to fit some MapReduce job.

## ACKNOWLEDGEMENTS

This book could not have been written without Dr. Jie Wu. Thank you Zachary Wagner for all the help along the way and for suffering through proof reading my writing.

Also thank you to Joshua Lloret for all your hard work and help along the way. Thank you for proof reading my papers, and helping me build and maintain our systems.

Thank you to Dr. Chiu Tan for all of your help along the way and for serving on my thesis deference committee.

Thank you Dr. Justin Shi for all your help with dealing with Eucalyptus and TCloud, and also for serving on my thesis deference committee.

Last but not least, Thank you to my family and friends for all your love and support along the way, especially these last two months.

For my Parents Raymond Blaisse and Linda Pasqua-Blaisse and Sister Mikaela  
Blaisse

# TABLE OF CONTENTS

<b>ABSTRACT</b> . . . . .	iii
<b>ACKNOWLEDGEMENTS</b> . . . . .	v
<b>DEDICATION</b> . . . . .	vi
<b>LIST OF FIGURES</b> . . . . .	x
<b>LIST OF TABLES</b> . . . . .	xi
<b>CHAPTER</b>	
<b>1. INTRODUCTION</b> . . . . .	1
1.1 Introduction . . . . .	1
1.2 MapReduce Structure . . . . .	2
1.3 Hadoop . . . . .	3
1.4 Schedulers . . . . .	4
1.5 Amazon EC2 . . . . .	5
1.6 Eucalyptus . . . . .	6
1.7 Structure of Thesis . . . . .	6
<b>2. LITERATURE REVIEW</b> . . . . .	13
2.1 Introduction . . . . .	13
2.2 Implementations . . . . .	13
2.2.1 Hadoop . . . . .	14
2.2.2 CouchDB . . . . .	14
2.2.3 Infinispan . . . . .	14
2.2.4 MongoDB . . . . .	15
2.2.5 Riak . . . . .	15
2.3 Related Papers . . . . .	15
2.3.1 Dynamic Proportional Share Scheduling in Hadoop . . . . .	15
2.3.2 Job Scheduling for multi-user MapReduce Clusters . . . . .	16
2.3.3 iShuffle: Improving Hadoop Performance with Shuffle-On-Write . . . . .	16
2.3.4 Online Load Balancing for MapReduce with Skewed Data Input . . . . .	16

2.3.5	Tarazu: Optimizing MapReduce on Heterogeneous Clusters . . . . .	16
2.3.6	Improving MapReduce Performance in Heterogeneous Environments . . . . .	17
2.3.7	Exploiting Cloud Heterogeneity for Optimized Cost Performance MapReduce Processing . . . . .	17
2.3.8	Optimizing Cost and Performance Trade-Offs for MapReduce Job Processing in the Cloud . . . . .	17
2.3.9	Evaluating MapReduce on Virtual Machines: the Hadoop Case . . . . .	18
2.3.10	A Performance Study on the VM Startup Time in the Cloud . . . . .	18
2.3.11	CLOUDLET: towards mapreduce implementation on virtual machines . . . . .	18
2.3.12	Scaling and scheduling to maximize application performance within budget constraints in cloud workflows	18
2.3.13	Towards optimizing hadoop provisioning in the cloud	19
<b>3.</b>	<b>APPROACHES TO SELECTION . . . . .</b>	<b>20</b>
3.1	Introduction . . . . .	20
3.2	Ratio Based Selection . . . . .	20
3.3	Best Fit Based Selection . . . . .	23
<b>4.</b>	<b>SYSTEMS . . . . .</b>	<b>28</b>
4.1	Introduction . . . . .	28
4.2	Net Cloud . . . . .	28
4.3	TCloud . . . . .	31
<b>5.</b>	<b>EXPERIMENTAL RESULTS . . . . .</b>	<b>33</b>
5.1	Introduction . . . . .	33
5.2	Physical Local Runs . . . . .	33
5.2.1	Ratio Based Selection . . . . .	35
5.2.2	Best Fit Selection . . . . .	35
5.3	Virtual Cluster Runs . . . . .	37
<b>6.</b>	<b>DISCUSSION OF RESULTS . . . . .</b>	<b>39</b>
6.1	Introduction . . . . .	39
6.2	Ratio Based Selection . . . . .	39
6.3	Best Fit Based Selection . . . . .	40
<b>7.</b>	<b>CONCLUSION . . . . .</b>	<b>43</b>

**BIBLIOGRAPHY . . . . . 49**

# LIST OF FIGURES

## Figure

1.1	The configuration of Map Reduce in a virtual environment . . . . .	3
3.1	Counters readily available to us through Hadoop . . . . .	21
4.1	The configuration of Map Reduce in a virtual environment . . . . .	30
4.2	The different levels of a virtual Hadoop cluster . . . . .	31
5.1	Ratio of shuffled bytes per CPU (ms) on a log-based 10 scale . . . . .	36

## LIST OF TABLES

### Table

5.1	Initial Runs information for 9 Hadoop Jobs . . . . .	34
5.2	Information about the different types of virtual machines. . . . .	36
5.3	Results of running the jobs on the two types of clusters . . . . .	37

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

With the increase in the availability of large sets of data, comes the need for better and more sophisticated methods of handling and processing these sets. Due to the size and complexity of these data sets, many users have moved to using distributed systems for storage and processing. With a distributed system, there are many different things that become much more complex and many more opportunities present themselves for issues. Out of this rose the paradigm of MapReduce.

The basic idea of MapReduce is to minimize the work of the programmer and remove a lot of the chances of creating an error because of the distributed computation. To do this all the work is either done in the Map Phase by the Map Tasks or in the Reduce Phase in the Reduce tasks. Communication and synchronization is taken care of by Map Reduce so that users are protected from misusing them.

This system has become quite popular with many different large companies adopting MapReduce. These large companies include, Facebook *Borthakur et al.* (2011), IBM IBM (2014), Yahoo! *OMalley* (2008), and Twitter *Weil* (2010). Due to its wide adoption, many different people have also looked at ways of speeding up Map Reduce.

Before talking about ways of speeding up MapReduce, it is important to understand how it works, and understand which implementation of the MapReduce paradigm is being used.

## 1.2 MapReduce Structure

The basic idea of MapReduce steams from Google's Paper in 2004 *Ghemawat and Dean* (2004). During the Map Phase, the input data is processed by many very small concurrent Map tasks. Then during the Reduce Phase, a much smaller number of much larger Reduce tasks takes the output from the many reduce tasks and combines it into the final desired answer. Perhaps the best way to visualize this is through an example.

Take the case where a user wants to count the number of occurrences of each unique word in a set of multiple books. In this case the books are the input data set. During the Map Phase, each small Map task takes as input a small piece of a book. In the Google paper they take these chunks to be 64 MB. Then each Map task takes each word within the chunk, and creates a key value pair (e.g. <the,1> <quick,1> <brown,1> <fox,1>). The key value pairs from many of these small map tasks would then be shuffled into a very small number, in this example 1, Reduce tasks. Say the input comes from 5 different Map tasks into 1 Reduce task.

Map 1 sends <the,1> <quick>

Map 2 sends <brown> <fox>

Map 3 sends <jumped,1> <over,1>

Map 4 sends <the,1> <lazy,1>

Map 5 sends <dog,1>

The Reduce task then takes these and produces:

<the,2> <quick> <brown> <fox> <jumped,1> <over,1> <lazy,1> <dog,1>

This is the end of the Map Reduce program and you can see that every unique word has been accounted for once and only once, and that the correct number occurrences has been found.

### 1.3 Hadoop

While this is a good high level view, there are many intricacies that are passed over, and are dependent on what implementation of MapReduce is used. For this paper will use Apaches MapReduce implementation named Hadoop *Hadoop* (2011). For Hadoop, the Map Phase is still simply made up of many small Map tasks. These task still take in a small chunk of the input data.

By default this is 64MB, but this is a configurable option so users can change this to meet their own needs. The Reduce phase in Hadoop does two things. The first is shuffle which takes the output put data from the Map task and shuffles them to the reduce tasks.

The second part is the reduce part. This cannot start complete until all of the Map tasks have completed. This part does what the Reduce phase does in MapReduce. These two parts together make up the Reduce task within Hadoop.

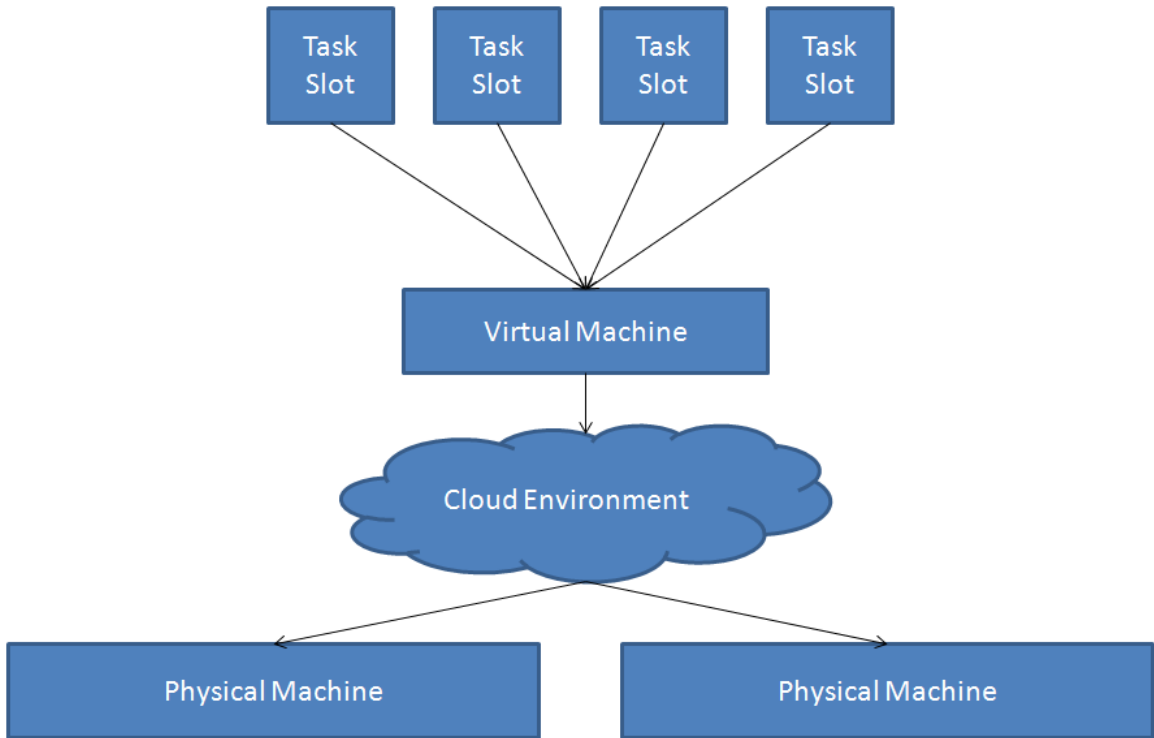


Figure 1.1: The configuration of Map Reduce in a virtual environment

## 1.4 Schedulers

For the remainder of this paper, Hadoop will be the implementation of MapReduce discussed. Now that it is understood how Hadoop implements Map Reduce, the focus becomes how it can be improved. Some papers have focused on how to improve the speed of Hadoop by focusing on the scheduling of the Map and Reduce tasks.

Hadoop itself comes with with three different build in schedulers, the fair share scheduler *fai* (2008), the capacity scheduler *cap* (2008), and the default FIFO scheduler.

A group from Hewlett-Packard's wrote a paper which proposed a Dynamic proportional share scheduler *Sandholm and Lai* (2010).

Another group of researchers from the University of California, Berkeley, Facebook, and Yahoo! proposed a method for job scheduling in a multi-user Map Reduce cluster *Zaharia et al.* (2009).

University of Colorado has a group which proposed a system called iShuffle, which they claim, improves Hadoops performance by doing Shuffle-On-Write *Guo et al.* (2013).

Another group comprised of member from Simon Fraser University, Indiana University Bloomington, and The Hong Kong Polytechnic University looked into doing online load balancing for Map Reduce when the data input is skewed *Le et al.*

Some people also looked into how to handle scheduling on machines which are not heterogeneous. Researchers from NEC Laboratories and Purdue University have proposed a system called Tarazu that aims to optimize Hadoop running on heterogeneous hardware *Ahmad et al.* (2012). A group from University of California, Berkeley, has also published a paper on improving the performance of Map Reduce in heterogeneous environments *Zaharia et al.* (2008).

More recently, research has started to look at how MapReduce behaves within a cloud environment, as well has how can the schedulers be changed to support cloud

computing. Very few papers, however, have focused on how to select Virtual Machines that best match a Map Reduce workload.

This paper will focus on using an environment similar to Amazon's EC2 cloud environment. This is illustrated in figure 1.1. The choice of Amazon comes from the fact that it is one of the largest and most widely used cloud computing systems. Amazon's EC2, allows users to spin-up virtual machines in their cloud at will.

Amazon EC2 has a few predefined sizes that have different costs that users select from when they are using the system. Eucalyptus, an open source cloud platform, is compatible with Amazon EC2. We will go about setting up our own cluster of servers running Eucalyptus so that we can have a comparable environment to Amazon to do our experiments.

## 1.5 Amazon EC2

Amazon EC2 is a pay as you go cloud environment *Amazon* (2010). The basic idea is that users rent some resources from Amazon's cloud for a certain rate. This rate is billed at a per hour basis, meaning, if a users only uses the resource for five minutes, then the users is still billed for an hour.

The most common resource that is rented from Amazon EC2 is virtual machines. Amazon offers many different sizes with different types of configuration. Some machines may be more specialized to handle CPU based jobs, while others might be optimized for memory or disk based jobs. Each of these different VM's comes with varying levels of CPU cores, RAM, and Storage capacity to match it's use. Each of theses virtual machines also has its own cost per hour.

This means that simply selecting the largest or strongest machine may not be the best option if one is trying to get the most for their money. The other resource that amazon offers is called elastic storage blocks. The basic idea of the the elastic storage blocks, is to offer a users more storage capacity to add to their virtual machine.

Like the virtual machines, storage space is has a per hour cost that depends on the amount of space used, as well as type of storage requested. These storage volumes, once purchased, can be attached to a virtual machine and used as extended storage capacity.

## 1.6 Eucalyptus

Eucalyptus is an open source implementation of a cloud computing cluster. Eucalyptus is written to be Amazon EC2 compatible. This means a few things for us.

The first, is that whatever we are able to run on a Eucalyptus, then it should also run on a Amazon EC2 virtual machine. This can be helpful, as it can allow users to test software on their own Eucalyptus clusters, before having to move to expensive Amazon EC2 machines.

Second, it means that we should get good approximations to results of running on the amazon EC2 cluster. This means that users can use their own clusters to test with fair confidence how software or systems may behave on Amazon's EC2.

The Eucalyptus software allows a system administrator to set it up to use either KVM or Zen. The choice often comes down to witch option is supported by the physical systems BIOS.

Very recently, Eucalyptus has been purchased by Hewlett Packard and has become HP Helion Eucalyptus. Later we will cover more about our Eucalyptus cluster.

## 1.7 Structure of Thesis

The remainder of this thesis will deal with the problem of selecting virtual machine to run in the cloud that will give the best performance for some MapReduce job. This means we will need to some ways of predicting what types of jobs will run best on

different types of virtual machines.

In chapter 2, we will do a review of literature surrounding the problem and the technology we plan to work on. This will include different types of software that is available, papers relating to MapReduce speed up, and some papers related to very similar issues as ours.

The first type of software that we will look into is different implementations of MapReduce. These will include, Hadoop, CouchDB, Infinispan, MongoDB, and Riak. Almost all of the softwares other than Hadoop use MapReduce, but do not allow users to directly run MapReduce jobs.

The papers that relate to MapReduce and Hadoop speed up will come next. These include, Dynamic Proportional Share Scheduling in Hadoop, Job Scheduling for multi-user MapReduce Clusters, iShuffle: Improving Hadoop Performance with Shuffle-On-Write, Online Load Balancing for MapReduce with Skewed Data Input, Tarazu: Optimizing MapReduce on Heterogeneous Clusters, and Improving MapReduce Performance in Heterogeneous Environments.

Next we come to papers that deal with problems that most similar to our own. These include, Exploiting Cloud Heterogeneity for Optimized Cost Performance MapReduce Processing, and Optimizing Cost and Performance Trade-Offs for MapReduce Job Processing in the Cloud.

The final set of papers will be papers about running MapReduce and Hadoop within the cloud. These include Evaluating MapReduce on Virtual Machines: the Hadoop Case, A Performance study on the VM Startup Time in the Cloud, CLOUDLET: towards MapReduce Implementation on Virtual Machines, Scaling and Scheduling to Maximize Application Performance Within Budget Constraints in Cloud WorkFlows, and Towards Optimizing Hadoop Provisioning in the Cloud.

This will be followed by chapter 3 which will be covering the Approaches to Selection that we will be looking at. There will be two different method of ours

which we will attempt to use throughout the paper to predict what type of virtual machine to use.

The first approach that we discuss in this paper will be the Ratio Based Selection approach. During the discussion of this approach we will cover many different things including, the intuition behind the approach, as well as the technical reason for this intuition.

We will also cover some of the metrics that are available to us from within Hadoop through the use of its HTML view. We will discuss what these different metrics are and how they may be helpful to us.

From there we will cover which metrics we first chose to use and why the original metrics did not work well. From there we will give the metrics that did work and explain why they were able to give us a good approximation for the necessary resources.

After this we move on to the the second approached in this thesis titled the Best Fit Based Selection approach. This will cover very similar things as the part discussing the other approach in this paper.

We will again talk about the different metrics and why they could be important to us. While the metrics will be the same ones available to us in the previous part, we bring them back up to go in more depth about some specifics.

Next we start to show how different metrics come together to describe different parts of the information that we need. From theses different parts, we show how to build the estimated that uses the information to give a value for each job. This value is used to select the virtual machine type for the cluster.

The next thing to follow is chapter 4. This chapter will focus primary on the different computing cluster that are used within the paper and why and how these cluster will be used within the paper.

The first system that we cover within this thesis is the Net Cloud computing

cluster. We cover as many different parts of this cluster as possible so that readers are hopefully able to use the information in this paper to test our results for themselves.

We will cover the hardware that makes up the cluster including the type of servers and their specifications, and the different types of switches that make up the cluster. All of the hardware is fairly simple commodity hardware.

After that, we will cover the different softwares that are used by this thesis that are available within the Net Cloud computing cluster. This includes the operating systems, the version of Hadoop and other softwares used.

Next we will cover the topology of the Net Cloud computing cluster. This is important to understand as it effects the different jobs by putting constraints on the network resources. You will find that our network setup is not overly complicated but still fairly interesting.

After this we move on to discuss the second computing cluster that we will use called TCloud. This cloud we will not have as much knowledge of the inner workings due to the fact that it is a public cloud available within Temples Computer Science department.

With that being said, we do attempt to first give as much information about the physical aspects of the TCloud computing cluster as possible. This comes out to be a general overview of the total power of the cloud cluster.

After this we move on to looking at the different softwares that we use and are available within the TCloud cloud computing cluster. This includes the operating systems, versions of Hadoop and other softwares that we use.

After that we cover the configurations of the three different types of virtual cluster that we will setup and use within the TCloud environment. These cluster represent a setup very similar to a setup you would find within Amazon EC2.

Then after we cover the setup of these three different virtual clusters, we will move on to discuss the general background network setup for the cloud computing cluster.

This does not help as much as the previous cluster as we don't know the location of the virtual machines.

The chapter that follows this is chapter 5 which presents the experimental results from running the multiple different jobs on the Net Cloud and the three virtual clusters within TCloud. This section present results from real runs.

The first part presented in this chapter covers the results from running jobs on real physical hardware. This means that the jobs where all run on the Net Cloud computing cluster. The results we get are the average value of the metrics for each of the jobs.

With theses values for the different metrics, we then look at how these metric values translate into values within the Ratio Based Selection approach. This means that each job gets a value for the ratio presented earlier in the paper.

After this, we use these values for the different metrics to get the different value for the different parts of the Best Fit Selection approach. Then we get the final value each job based on this metric. These values along with the values for the Best Fit Selection approach will be discussed later in the paper.

The next thing that is coved in this chapter, is the results from running on the three different types of clusters within TCloud that are made up of different size virtual machines. These results are used to represent times similar to those we would find in Amazons EC2.

All of the results that are presented from this part are times that given in seconds. This is simply used to show how fast jobs acctuly run when run on the different types of clusters that vary in type of virtual machine. We ran every job we used in the paper on all three types of virtual machines.

In this section we also quickly look at and state which of the three different clusters are the best best on the which of the three clusters ran the fastest. We do not go into why this may be or what this means for our two different approaches until later in

the paper.

The chapter that comes next is chapter 6 titled Discussion of Results. In this chapter we will look back at the previous chapters in the paper and see what everything means for our attempts to solve the problem of selecting virtual machines based on the MapReduce job that is being run.

The first thing done in this chapter is to look at how the Ratio Based Selection approach holds up with the results from the Net Cloud cluster and the result from the three virtual cluster of different size virtual machines inside of TCloud.

First we cover what the different ratio values mean for the Ratio Based Selection approach. This means looking at each job individually and say which of the virtual clusters of different size virtual machines within TCloud should be best for it.

After we present the results of the prediction based on results from running on Net Cloud and applying the Ratio Based Selection approach, we then look at which of the virtual clusters of different size virtual machine within TCloud actually ran best.

Next we look at the prediction that is based on the Ratio Based Selection approach, and the results of finding out which type actually ran best and we compare them to see how this approach did in our experimental setup.

After looking at how our approach did on correctness, we then move on to looking at how our approach did at predicting the benefit from the different jobs for running on the machines that they were predicted to run best.

Following this we will look at how the Best Fit Based selection approach works when it is applied to the result from the Net Cloud computing cluster and the three virtual cluster of different size virtual machines within TCloud.

We review the different values that gained from applying the results from Net Cloud to the Best Fit Based Selection approach. Then we list which of the three virtual cluster of different size virtual machine within TCloud should be best for each

of the jobs.

We next look at the how the results from running on the Three TCloud virtual cluster compare to our predictions. We then look at how correct our method was at predicting the jobs preferred cluster, and how accurate we are at predicting the benefit gained from running on it.

The final chapter, chapter 7 titled conclusion is exactly as you would expect it to be. It is a summarization of everything that covered in this paper and what it means for our different results.

## CHAPTER 2

# LITERATURE REVIEW

### 2.1 Introduction

In this chapter we will look at different works that are related to our research. We will also do a quick review of different methods that have been proposed for dealing with Selection of Virtual Machines as they relate to MapReduce and Hadoop.

The programming paradigm of MapReduce has become quite popular since its introduction. Due to its popularity and multitude of uses, MapReduce , have found ways to adapt MapReduce into their current operations. These companies include the likes of Facebook *Borthakur et al.* (2011), IBM *Zikopoulos et al.* (2012), Yahoo! *OMalley* (2008), and Twitter *Weil* (2010). In subsection 2.2 we will cover the different types of implementations of MapReduce. After this, section 2.3 will cover the few paper that consider a similar issue.

### 2.2 Implementations

Map Reduce is simply a programming paradigm, and for this reason there are many different implementations available to users. We will cover the major iterations in the following subsections. We will cover Hadoop in 2.2.1, then CouchDB in 2.2.2, followed by Infinispan in 2.2.3, next is MongoDB in 2.2.4, and finally Riak in 2.2.5.

### **2.2.1 Hadoop**

The first implementation, Apache's Hadoop *Bialecki et al. (2005)*, may be the most commonly used distribution of MapReduce. Hadoop is an open source implementation of MapReduce created by the Apache Software Foundation. It is implemented using Java. Hadoop works on Windows, Linux and Mac OS systems. The software is written to create a Hadoop cluster across a large number of machines. These machines will then run the HDFS file system and Map Reduce jobs.

In addition to Hadoop, there are other Apache projects which are built to extend the applicabilities of MapReduce by using Hadoop. These include, Ambari, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Spark, Tez, ZooKeeper, and other software packages. These pieces of software offer things ranging from monitoring of Hadoop clusters, to data serialization, to extending the compute engine for Hadoop.

### **2.2.2 CouchDB**

CouchDB is a database built by the Apache Software Foundation. It uses JSON to deal with documents, and HTTP to deal with its API's. It uses MapReduce implemented in JavaScript to handle indexes. While it is not a full MapReduce environment, it is worth mentioning. It has distributions available for on Windows, Mac OS and Linux.

### **2.2.3 Infinispan**

Infinispan is a software developed by Red Hat. It stores Key and Value information. The software is written in Java and is Open Source. It is not directly a MapReduce distribution, but it does have MapReduce built in to its API. There are versions of Infinispan available for Linux, Windows, and Mac OS.

#### **2.2.4 MongoDB**

MongoDB is a NoSQL data base that describes itself as a document database. It is open source and written in C++. The software uses MapReduce to try and condense large amounts of data into smaller aggregated results. The Map Reduce is implemented through there own JavaScript functions.

#### **2.2.5 Riak**

Riak is a Key value NoSQL data store. It is available for Linux, and Mac OS consisting of an open source version and an enterprise version. MapReduce is used in the software to do queries on larger data sets that are non-key-based.

### **2.3 Related Papers**

There are a surprising small number of papers that directly deal with the issue discussed in this thesis. Due to this the papers that will be presented will mostly look at smaller issues that are related to issues in this thesis.

This will include papers looking at how to do scheduling within MapReduce. Their will be papers that look at virtual machine selection for other applications and in other environments. There will also be a small number of papers that will look at the issue being dealt with in this paper.

#### **2.3.1 Dynamic Proportional Share Scheduling in Hadoop**

A group from Hewlett-Packard's wrote a paper, which proposed a Dynamic proportional share scheduler *Sandholm and Lai* (2010). The authors claim that it would allow users to control the amount of resources used by adjusting their spending. They offer the design and how they implemented it. They then present how it works in practice.

### **2.3.2 Job Scheduling for multi-user MapReduce Clusters**

Another group of researchers from the University of California, Berkeley, Facebook, and Yahoo! proposed a method for job scheduling in a multi-user Map Reduce cluster *Zaharia et al. (2009)*. They discuss issues that make scheduling in MapReduce difficult, and they talk about creating their own fair share scheduler used at Facebook. They specifically talk about environments with multiple users.

### **2.3.3 iShuffle: Improving Hadoop Performance with Shuffle-On-Write**

The University of Colorado also has a group, which proposed a system called iShuffle, which they claim, improves Hadoop's performance by doing Shuffle-On-Write *Guo et al. (2013)*. The authors say that the shuffle phase significantly affects the running time of MapReduce Jobs. They then implement and present iShuffle which is does shuffle on write.

### **2.3.4 Online Load Balancing for MapReduce with Skewed Data Input**

Another group comprised of members from Simon Fraser University, Indiana University Bloomington, and The Hong Kong Polytechnic University looked into doing online load balancing for Map Reduce when the data input is skewed *Le et al.*. The authors do not implement a load balancing schema within in Hadoop, but rather show that it could serve as a good idea for others to implement.

### **2.3.5 Tarazu: Optimizing MapReduce on Heterogeneous Clusters**

Researchers from NEC Laboratories and Purdue University have proposed a system called Tarazu that aims to optimize Hadoop running on heterogeneous hardware *Ahmad et al. (2012)*. The authors list some things that they claim are the major issues with running on MapReduce on heterogeneous machines. They also offer Tarazu which improves Hadoop on heterogeneous hardware.

### **2.3.6 Improving MapReduce Performance in Heterogeneous Environments**

A group from University of California, Berkeley, has also published a paper on improving the performance of Map Reduce in heterogeneous environments *Zaharia et al.* (2008). They present some drawbacks from the traditional Hadoop schedulers when running on heterogeneous machines. They then present their own scheduler called the LATE scheduler, which they say, improves scheduling on heterogeneous machines.

### **2.3.7 Exploiting Cloud Heterogeneity for Optimized Cost Performance MapReduce Processing**

The first paper is *Zhang et al.* (2014a). This paper also talks about the measurable difference for jobs run on different clusters types. The authors of this paper decide to look at workloads of multiple Hadoop jobs rather than just one type of job. The authors look at using a simulation-based framework to do selection of Hadoop clusters. They then introduce their ranking system they call PScore to evaluate Hadoop jobs. They consider that, the clusters cost the same amount per hour, the time necessary to run the jobs can vary. Therefore, it costs more to run on a bad cluster because it must use another time unit. These effects are compared on several different workloads.

### **2.3.8 Optimizing Cost and Performance Trade-Offs for MapReduce Job Processing in the Cloud**

The second paper is *Zhang et al.* (2014b), conducted by the same group as the previous paper. In this paper, they look at how they should split their budget to create multiple smaller clusters of different types, depending on the jobs in a workload. They again use their PScore measurement.

### **2.3.9 Evaluating MapReduce on Virtual Machines: the Hadoop Case**

A group from Huazhong University of Science & Technology, and China Development Bank, presented their work in *Ibrahim et al. (2009b)*. They propose using MapReduce, and more specifically, Hadoop within a Cloud environment. They then present their results from running Hadoop on virtual machines. They also present issues that continue to exist with this system.

### **2.3.10 A Performance Study on the VM Startup Time in the Cloud**

Another group from the University of Virginia presented a performance study on the time it takes a virtual machine to start in *Mao and Humphrey (2012)*. What they showed was that the startup time for virtual machines could vary greatly depending on operating system, virtual machines size, and virtual machine provider.

### **2.3.11 CLOUDLET: towards mapreduce implementation on virtual machines**

In *Ibrahim et al. (2009a)*, the group from Huazhong University and China Development Bank give an overview of their work implementing Map Reduce on virtual machines. They then compare the jobs run on physical machines to jobs run on virtual machines.

### **2.3.12 Scaling and scheduling to maximize application performance within budget constraints in cloud workflows**

Still another group from the University of Virginia present work in *Mao and Humphrey (2013)*. The authors of this paper do not look at MapReduce jobs specifically, but rather at maximizing general application performance in the cloud while working within a budget constraint.

### 2.3.13 Towards optimizing hadoop provisioning in the cloud

Finally, a group from IBM Research Almaden and Purdue University worked on optimizing Hadoop Provisioning in the Cloud in *Kambatla et al. (2009)*. They show why MapReduce can work well with the cloud framework. They then present a preliminary method for improving MapReduce provisioning. They do this by analyzing and comparing the resource consumption of applications, and then comparing them to a database of known workloads.

## CHAPTER 3

# APPROACHES TO SELECTION

### 3.1 Introduction

This paper will present two different approaches to selecting virtual machines based on the Hadoop MapReduce jobs that a user wants to run. First in Section 3.2 and Section 3.3, the two different proposed selection process will be introduced. Then in Chapter 5 we will cover how the two approaches perform in our test beds. Later in chapter 6 we will cover how the two approaches hold up to and how well they performed overall.

### 3.2 Ratio Based Selection

The primary issue we look at is cluster configuration. We would still like to select as good a configuration as possible for our Hadoop cluster, but we do not want to do this by trying every possible configuration.

A notable fact about Hadoop jobs is that, usually, a job will run multiple times, because it runs through some entity. For example, Google parses the log files using their version of MapReduce. Another example would be someone sorting large amounts of data from some tests.

While each run will not have the same input data, each run will still be running the same program, which should have very similar characteristics to another run of the same program. This gives us the ability to trust in the characteristics from a smaller set of data to predict the characteristics of larger data sets.

There will be several characteristics that we believe are key. These values are simply counters that already exist within the Hadoop framework. This means that we are able to easily access any of the metrics without much difficulty. An example of these characteristics can be seen in figure 3.1

<b>File Input Format Counters</b>	
Bytes Read	118
<b>FileSystemCounters</b>	
HDFS_BYTES_READ	240
FILE_BYTES_WRITTEN	54,372
<b>Map-Reduce Framework</b>	
Map output materialized bytes	28
Map input records	1
Spilled Records	2
Map output bytes	18
Total committed heap usage (bytes)	173,539,328
CPU time spent (ms)	210
Map input bytes	24
SPLIT_RAW_BYTES	122
Combine input records	0
Combine output records	0
Physical memory (bytes) snapshot	190,488,576
Virtual memory (bytes) snapshot	846,843,904
Map output records	2

Figure 3.1: Counters readily available to us through Hadoop

The first metric we look at is CPU Time. This value is simply how much time the different tasks spend on the CPU, measured in milliseconds. This metric will give us an idea of how much time the task spent on the CPU doing work, rather than waiting for data, and I/O reads and writes. We will label this value as  $\phi$  for the remainder of this paper.

Another metric that we looked at using was the total Number of HDFS bytes. In Hadoop, there are two counters for this process; they are HDFS read, and HDFS bytes written. This would give us an idea of how much I/O is occurring.

Our original thought was to use  $\phi$  and the HDFS bytes to create a metric for whether an application is I/O-heavy or CPU-heavy. This, unfortunately, did not

work as a good indicator in practice when we tested it.

The next metric we looked at using was Reduce shuffle bytes, which we label  $\gamma$ . The Hadoop framework gives us Reduce shuffle bytes directly as a counter. This is a measure of how much data passes from all of the Map tasks to the Reduce tasks in the Shuffle stage.

This is very helpful to use, since a large amount of congestion can, and often does occur when data transfers over the network during the shuffle phase. The network is really not taxed to heavily by MapReduce during any of the other phases.

This is due to the fact that many Map Tasks can be finishing around the same time. This can cause a large amount of delays for the Reduce stage. Some of this problem can be mitigated by doing early start on the reduce tasks. Using this metric by itself is not enough.

The simple problem with using only this metric is that by just adding more data to process, this value would raise. To offset this issue, we decided to revisit the idea of a ratio. This time we use  $\gamma$  and  $\phi$ .

This will give us the amount of data shuffled per millisecond spent on the CPU. This gives us the amount of data per actual unit of work done. This can also be seen as how much more data it transfers than it spends on the CPU. Theoretically, if we know the speed of our connections, we can estimate how long it should take us to transfer the data.

Using this metric, we found that Word Count had a ratio of 117.8997, PI had a ratio of 0.1109, Pentomino had a ratio of 0.0194, TeraSort had a ratio of 9125.2584, TeraGen had a ratio of 0, Grep had 0.0148 as a ratio, MRBench's ratio was 0.0089, DFSIOTest Read had a ratio of 0.0464, and DFSIOTest Write had a ratio of 0.0380. A graph with TeraSort's, and Word Count's Ratios removed, can be found in Figure 5.1. This will be expanded on more in Chapter 5 and we will discuss what this means and which machines work best based on the results.

We will give 2 examples quickly of this selection method. These should help illustrate how this method classifies Map Reduce jobs.

For the first example assume that some job has a shuffle bytes value of 215052852, and a CPU Time value of 1824033. Then the Ratio value of Shuffle bytes divided by CPU Time would be 117.899649841861413691528. Since this value is greater than 1, the job is classified as a I/O bound job that should prefer a few Larger type machines.

For the second example, assume that some job has a shuffle bytes value of 23400, and a CPU Time value of 1582036. Then the Ratio value of Shuffle bytes divided by CPU Time would be 0.0147910667. Since this value is less than 1, the job is classified as a CPU bound job that should prefer many smaller type machines.

We will also show you later in chapter 5 that covers results where and how we got these values. Then in chapter 6 we will discuss how this approach fairs when put to the test of real data.

### **3.3 Best Fit Based Selection**

As said previously, there are a few different metrics Hadoop offers by default. These metrics are Bytes Written, File Bytes Read, HDFS Bytes Read, File Bytes Written, HDFS Bytes Written, Bytes Read, Shuffle Bytes, and CPU Time.

The Jobs that we chose to test with were Word Count, PI, Pentomino, Word Count TeraSort, TeraGen, and Grep. All of these metrics for these jobs are data that is readily available through Hadoop. In this section, we will cover how the best fit based selection works before offering experimental results from our testbeds in Chapter 5.

We can see that, while some map tasks may use more resources and some map tasks may use less, it is safe to assume that these are the values for map tasks of a specific job. This is due to the fact that even if the input data being more "difficult", the job still should have the same very small input data size.

A simple and naive approach for selecting virtual machines would be to match MapReduce jobs to machines, by select a single metric that we uses to fit machines regardless of what job we are matching.

This paper will instead still use the idea of fitting jobs to virtual machines based on metrics, however we will considers all six of the metrics simultaneously when making the decision. We will show that this gives us a good deal of accuracy when predicting which machine fits best.

HDFS Bytes Read and HDFS Bytes Written can together be considered I/O done. This is due to the fact that for Hadoop MapReduce, all I/O is done to the HDFS file system. So, bytes read and written would be all the I/O to the HDFS file system.

For memory usage, there are three possible metrics we could use. Looking at the three metrics, Physical Memory appears to give the most useful information for selecting virtual machines, as it gives the amount of real physical memory necessary to fit the jobs. This could be due to the fact that this represents how much real memory is used on the physical system.

Finally, the CPU Time metric provides us with a good estimate of CPU usage. This is different from simply timing a jobs, as the CPU Time is only the time accumulated on the CPU itself. This means that time spent waiting for I/O, waiting to get onto the CPU, and other things do not affect this value.

The three parts that will be used to make up the selection process that has been decided on, and we now move on to how they are used. The HDFS Bytes Read and HDFS Bytes Written will be summed together to make up  $Req_{I/O}$ . Physical Memory will be simply used as is, and will be called  $Req_{Memory}$ .

For CPU Time, we decide not to use for prediction at this time. CPU Time will simply be used as the CPU Time for prediction. This means it will not be used to decide the machines, but rather, it will be used to determine how long the application will take once the machine type has been selected.

When looking at  $Required_{I/O}$ , we will use equation 3.3-1 to determine the effectiveness of a virtual machine to fulfill a requirement for a Map task.

$$\frac{VM_{I/O} - Req_{I/O}}{10 * VM_{I/O}} \quad (3.3-1)$$

This equation will give us the amount of I/O wasted proportional to the total amount of I/O available by using a particular type of virtual machine.

The reason for multiplying the denominator by 10 is simple. The entire system I/O cannot be used for the MapReduce jobs. For this reason, we multiply the denominator by 10 to provide the necessary room for the system.

We will use a similar type of equation for Memory. This is show in equation 3.3-2.

$$\frac{VM_{Memory} - Req_{Memory}}{VM_{Memory}} \quad (3.3-2)$$

Just like for I/O, this gives us the proportion of wasted Memory to Memory available. With both equation 3.3-1 and equation 3.3-2, a negative value would mean that the virtual machine does not have enough of the resource to fulfill the task requirements. We do not need to give any multipliers for the denominator for this metric.

CPU Time is a little trickier to factor in. One of the hard problems is that most systems for spinning up virtual machines only offer you more cores, and not more speed as an option. There are some specific options available through Amazon EC2 that offer more powerful CPUs, but for this paper, we will not use CPU Time for prediction purposes.

It would not be too difficult to see how it would fit in if we moved to a system like Amazon EC2 which supports different speeds for CPU. The ratio for it would be similar to the previous two. Later when we combine the previous two metrics, you can easily see how it would fit in.

Now we need an equation that will combine equation 3.3-1 and equation 3.3-2 to

create the overall equation for a system's usefulness. This is shown in equation 3.3-3.

$$\min\left(\frac{VM_{I/O} - Req_{I/O}}{10 * VM_{I/O}}, \frac{VM_{Memory} - Req_{Memory}}{VM_{Memory}}\right) \quad (3.3-3)$$

The idea is to find the virtual machine that has the smallest ratio for either metric. It is important to note that if the value is negative, then we do not consider the specific type of virtual machine as a viable option.

Usually, there are a small number of options for virtual machines. This means that for each variation of virtual machine, we simply find its value using equation 3.3-3. The cost for doing this by a computer is negligible meaning we do not need to worry about doing it.

We then store the value for all the variations. For simplicity, we will simply say  $VMValue[k]$  gives the value from equation 3.3-3 for virtual machine configuration  $k$ . This means we could look at this as equation 3.3-4 :

$$\min(\min(VMValue[k])) = \min\left(\min\left(\frac{VM_{I/O} - Req_{I/O}[k]}{10 * VM_{I/O}}, \frac{VM_{Memory} - Req_{Memory}[k]}{VM_{Memory}}\right)\right) \quad (3.3-4)$$

We will quickly give two examples to try and illustrate this system works for selecting virtual machines based on Map Reduce jobs. Hopefully this will make the selection process a bit clearer to the reader.

First Assume some jobs has and I/O of value of 3,669,042.6 and a physical memory value of 196,680,908.80. Then assume the smallest machine type has a of  $1e+9$  bytes available for I/O and 1024000000 of memory available for use. Then the I/O proportional value for the small machine is 0.09996330957 and the memory proportional value is 0.8079288, meaning the machine value is 0.09996330957. This means the job should use the smallest possible virtual machine type since this value is positive and for the smallest possible machine type.

Second Assume some jobs has and I/O of value of 1,000,000,017 and a physical

26 memory value of 21,313,126.40. Then assume the smallest machine type has a of  $1e+9$  bytes available for I/O and 1024000000 of memory available for use. Then the I/O proportional value for the small machine is  $-1.7e-9$  and the memory proportional value is 0.9791864, meaning the machine value is  $-1.7e-9$  . Since the value is negative, we would repeat this again using the next size of virtual machine. This would continue until we find a machine with a positive value.

## CHAPTER 4

# SYSTEMS

### 4.1 Introduction

In the following two sections we will present the two main systems we used for testing our strategies. We will cover how the two systems are constructed as well as how each of the two systems are put to use for selection. For one of the systems that we setup, we will also share our experiences from setting up and installing it.

First, in section 4.2 we will introduce the Net Cloud cluster. We will give a breakdown of the hardware that makes up this cluster, we will cover the software that is running on the system, the topology of the cluster, and some of our experiences of setting up , working with and maintaining the cluster.

The next section, is section 4.3 which will cover the TCloud cluster. In this section we will cover the hardware that makes up the cluster, the softwares that run the cluster, the topology of the TCloud cluster, and the three different virtual cluster of different size virtual machine within TCloud.

### 4.2 Net Cloud

The Net Cloud test bed is the system used for initial testing. The cluster is made up of 32 servers. Each system is a Dell PowerEdge R210. Each server has an Intel Pentium G2120 dual core processor that runs at 3.10GHz. Each system contains 4 GB or RAM running at 1600 MHz. Each system has a 500 GB HDD, and a Gigabit Ethernet connection.

The servers are interconnected in a tree like fashion, which can be seen in figure 4.2. Each of the switches used for interconnection are Cisco small business switches. Each switch contains 10 Gigabit Ethernet ports. More about this can be found in our paper *Blaisse et al. (2014)*.

The Net Cloud cluster allows us to gain information about Map Reduce jobs in a cluster of physical machines. For selection, both methods use the information garnished from running jobs on the Net Cloud cluster to then predict which virtual machines would be best. The data from the run on Net Cloud will be presented later in chapter 5.

We actually built and setup the Net Cloud cluster. I will now share some of my experience and describe how we went about constructing the cluster. The Net Cloud cluster is housed inside of Temple University's SERC data center.

The first step was to install the hardware into the server racks. Our cluster took up two server racks inside of the SERC data center. The servers are setup in groups of 4 with a Cisco switch they are wired into above them. We put 4 of these groups into reach of our server racks. At the top of each server rack is a switch that is connected to the switches from the 4 groups.

In the first server rack, we have some other equipment needed to run the cluster. At the very top of the first server racks is a switch that connects to the switches at the top of the two server racks. Under this is a server that has the job of providing NAT, DNS, and DHCP to the rest of the cluster. This server is connected directly to the previously mentioned top switch.

As an operating system, each server is configured to run CentOS 6.6. We use Hadoop as our MapReduce software. We run Hadoop version 1.2.1. For Java, we use OpenJDK. Our version of OpenJDK is 1.7 sixty-four bit.

Since we got to build this system, and are constantly looking into how to improve this system, we will also cover some future improvements that are either planned or

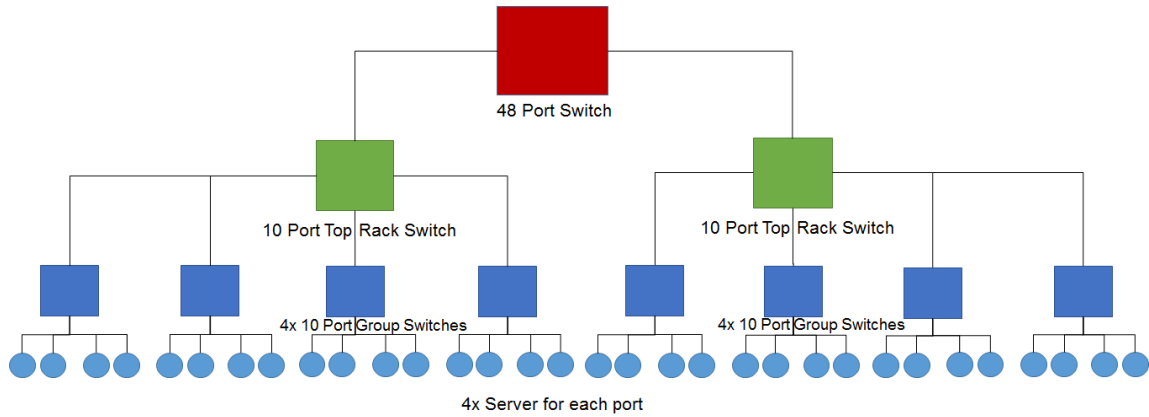


Figure 4.1: The configuration of Map Reduce in a virtual environment

are in the process of being done.

One improvement that we have been able to make since the experiment where run, was the installation of the Eucalyptus software suite into the Net Cloud Cluster. While this would not have helped with either approach that is presented in this paper, it does offer us more of an ability for later papers to see what is happening from the providers stand point.

Another improvement that is in the process of being doing the Net Cloud cluster, is the instillation of Pica8 Open Flow switches into the cluster. This will change the topology of the cluster slightly. In the future, we hope that this improvement might allow us to expand our approaches for speeding up Hadoop into the software defined network space.

We can also some interesting point for those who might be trying to create their own Physical Hadoop cluster. The first observation is that it can be quite tricky to setup your own NAT, DNS, and DHCP for the cluster so that Hadoop and or Eucalyptus works Properly. For the most part if their was an issue with either software, it ended up being caused by a networking problem

The Topology of our Net Cloud computing cluster will be changing with the instillation of the Pica8 Open Flow switches. The cluster will still be split into groups

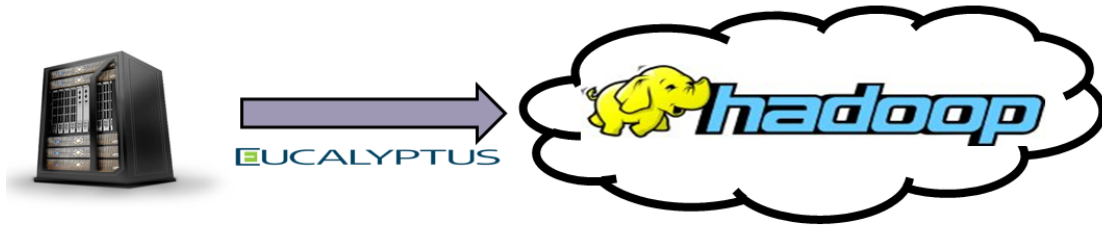


Figure 4.2: The different levels of a virtual Hadoop cluster

of 4 servers which connect into a group switch.

From there, each group switch will wire into each of the three Pica8 Open Flow switches. The path from one machine weather it be virtual or physical, will depend on the setup of the open flow setup.

### 4.3 TCloud

The TCloud test bed on is the second system and it is used for testing jobs in a real cloud environment. TCloud is a private cloud-computing cluster at Temple University TCl. The cluster is made up of 12 Dell Power Edge R614 Servers, which offers 96 conventional CPU Cores. Each physical sever has 128GB of RAM. The servers are connected with 4-way redundant 10 GB Ethernet.

They also have a 2-way redundant InfiniBand. TCloud currently runs on Eucalyptus 3.3 to offer an AWS-compatible environment. Since TCloud runs Eucalyptus, it allows us to spin up many Virtual Machines of varying configurations. Next, we will describe the Hadoop cluster and the virtual machines used to create each type.

There are three different clusters that are used for testing. The three cluster are the Small cluster, the Medium cluster and the Large cluster. The Ratio Based Selection method described in subsection 3.2 will only use the Small cluster and Large cluster. The Best Fit Based selection method described in subsection 3.3 will uses all the different sizes of virtual machines.

The Larger cluster consists of two virtual machines. In Eucalyptus these machines are of type, m2.4xlarge. This means that each machine has eight CPU cores. Each machine has 4096 MB or 4 GB of RAM. Finally each machine has 60 GB of HDD storage to use.

The Medium cluster consists of four virtual machines. The machine types in Eucalyptus is m3.xlarge. Each machine has 4 CPU cores. Along with that each machine has 2048 MB or 2 GB of RAM. For HDD storage space, each machine has 15 GB of storage.

Finally, the Small cluster consists of eight virtual machines. The Eucalyptus machine type would be m1.xlarge. For CPU cores, each machine has 2. Each machine has 1024 MB or 1 GB of RAM. The HDD storage space for each is 10 GB.

Since the machines are virtual, we have no idea what kind of topology exists between each type of virtual machine. Each virtual machine is running CentOS version 6. Each cluster Hadoop cluster is constructed using Hadoop version 1.2.1 sixty-four bit.

For the rest of the paper we will refer to the cluster that is made up of two m2.4xlarge machines as the Larger cluster. The cluster with four m3.xlarge virtual machines will be called the Medium cluster. Finally the cluster with eight m1.xlarge machines will be called the Small Cluster.

## CHAPTER 5

# EXPERIMENTAL RESULTS

### 5.1 Introduction

In this chapter, we will present our results from testing our two methods in chapter 3 on our two test beds presented in chapter 4. We will also discuss what the results from the runs mean about the effectiveness of our proposed methods. Testing can be seen as two different parts.

The first presented in section 5.2 is the initial runs with a small data set to gain information about the characteristics of a job. The second presented in section 5.3 will give us the results from running in the virtual clusters. Finally, in Chapter 6 we will discuss these results.

### 5.2 Physical Local Runs

This section will cover the results from running the different jobs on the physical machines. To get the values, we run each job with a small subset of the data that we would use for the different jobs on the virtual machines. The values are the average of 10 different Map tasks counters selected at random. These results can be seen in table 5.2.

As you can see, different jobs have very different values for different characteristics. It is interesting to note that some metrics that we have previously stated as being important for prediction, have wide disparities between the different jobs. As an example look at HDFS Bytes Written. The job with the least would be Pentomino

Table 5.1: Initial Runs information for 9 Hadoop Jobs

Metric	Word Count	PI	Pentomino	TeraSort	TeraGen
Bytes Written	11302255	97	0	3000000000	15000000000
File Bytes Read	315284416	33012	16.2	6071852754	0
HDFS Bytes Read	637467252	360390	7590434	3000184732	170
File Bytes Written	570661132	82745869	110228909.2	9134491993	109109.6
HDFS Bytes Written	11302255	215	0	3000000000	15000000000
Bytes Read	637381508	177000	8149038	3000180224	0
Shuffle Bytes	215052852	42000	12006	3060000276	0
CPU Time (MS)	1824033	378626	617714	335333	244803
Metric	Grep	MRBench	DFSCIOTest Read	DFSCIOTest write	
Bytes Written	3818	3	70.3	70.4	
File Bytes Read	191983	13	86.3	86.4	
HDFS Bytes Read	637460630	247	954418	235	
File Bytes Written	43503497.9	165595.1	110020.6	110024.8	
HDFS Bytes Written	3818	3	70.3	1048646.4	
Bytes Read	637383922	3	112	112	
Shuffle Bytes	23400	19	86	86	
CPU Time (MS)	1582036	2145	1852	2261	

with 0, while TeraGen had the most with a value of 15000000000.

Using this information, you can see which jobs should be best on which machines. Subsection 5.2.1 will cover the selection based on the method proposed in chapter 3. Then Subsection 5.2.2 will cover the results in the scenario of the Best Fit selection process presented in chapter 3.

### **5.2.1 Ratio Based Selection**

In this subsection we will show how to use the results to predict the type of machine to use. We found that Word Count had a ratio of 117.8997, PI had a ratio of 0.1109, Pentomino had a ratio of 0.0194, TeraSort had a ratio of 9125.2584, TeraGen had a ratio of 0, Grep had 0.0148 as a ratio, MRBench's ratio was 0.0089, DFSIOCTest Read had a ratio of 0.0464, and DFSIOCTest Write had a ratio of 0.0380. This can be seen in figure 5.1

Using this information, we can see that Word Count is predicted to work best on the Larger machine cluster. PI looks like it would be best run on the Small virtual machine cluster type. Looking at Pentomino, it shows that the Small machine cluster type is the best recommended cluster.

TeraGen looks like it would work best on the Larger virtual machine type cluster. It is predicted that Grep works best on Small virtual machine type cluster. MRBench again looks to run best one Small virtual machine type cluster. Finally, DFCIOCTest read and write are predicted to run best on the Small virtual machine type clusters.

### **5.2.2 Best Fit Selection**

In this subsection, we will present how the Best Fit Selection presented in 3 recommends for us to select virtual machine clusters. As you can see, the minimal value for PI is 0.8202972 on small type machines.

Pentomino preferred a small type of virtual machines, with a minimal value of

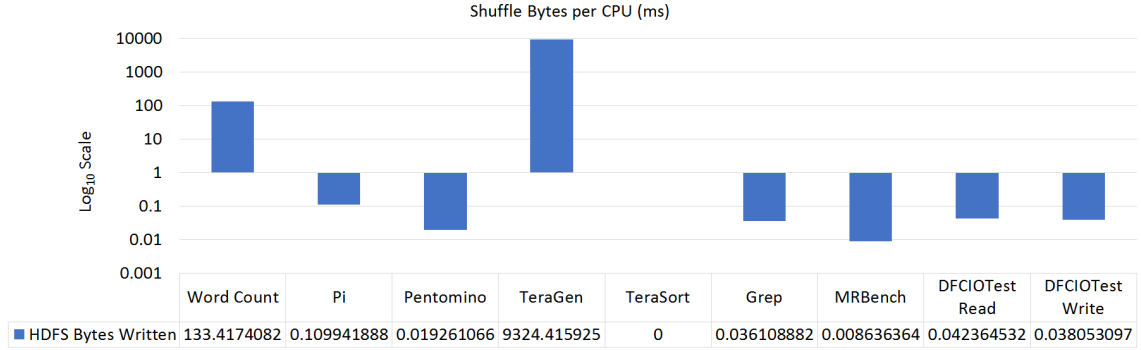


Figure 5.1: Ratio of shuffled bytes per CPU (ms) on a log-based 10 scale

Table 5.2: Information about the different types of virtual machines.

Jobs	Small ( m1.xlarge )	Medium ( m3.xlarge )	Large ( m2.4xlarge )
PI	0.8202972	0.9101486	0.9550743
Pentomino	0.8146544	0.9073272	0.9536636
WordCount	0.8079288	0.9039644	0.9519822
TeraSort	0.795928507	0.863952338	0.9534914
TeraGen	-0.000000017	0.333333322	0.833333331
Grep	0.1091744	0.5545872	0.7772936

0.8146544. Word Count had a minimal value of 0.8079288, meaning small type machines are the best. I should note that TeraGen had a negative value for the Small virtual machine cluster type.

TeraSort preferred a small type of virtual machines, with a minimal value of 0.795928507. TeraGen had a minimal value of 0.333333322, meaning medium type machines are the best. Finally, Grep preferred small type machines with a minimal value of 0.1091744.

The full different values can be seen in table 5.2. This means that the next largest non-negative, which as previously mentioned is Medium, would be the best. This the only job for which we tested that ended up being predicted as not a small size machine.

Table 5.3: Results of running the jobs on the two types of clusters

Job	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
PI (L)	21.161	21.06	22.136	21.428	22.207	20.89	21.984	21.204	21.036	20.97
PI (M)	25.634	24.243	24.518	25.382	24.356	24.722	24.414	24.666	24.313	25.157
PI (S)	19.501	18.227	19.29	19.276	19.399	19.708	18.489	19.4	19.324	18.593
Pentomino (L)	428.516	438.237	433.485	434.214	436.834	439.065	427.175	432.041	436.636	434
Pentomino (M)	979.947	976.659	974.575	976.626	979.798	985.868	982.622	975.409	977.325	979.799
Pentomino (S)	414.466	414.381	414.197	418.525	414.323	414.218	414.194	414.66	419.231	419.671
WordCount (L)	577.46	554.086	538.351	552.307	530.661	530.487	588.586	533.571	535.493	526.639
WordCount (M)	1226.303	1220.649	1223.581	1218.396	1221.854	1220.751	1220.394	1216.915	1220.635	1222.607
WordCount (S)	507.345	509.859	505.958	505.198	504.853	506.881	505.255	506.726	507.499	507.072
TeraSort (L)	107.114	142.347	152.3	160.482	161.436	160.216	186.631	186.106	207.02	171.653
TeraSort (M)	148.781	158.664	141.469	185.44	176.074	156.402	181.574	149.197	171.424	192.14
TeraSort (S)	76.938	84.464	77.72	87.553	86.991	88.238	89.756	85.267	82.116	89.229
TeraGen (L)	39.992	52.742	36.977	33.905	32.891	24.653	25.872	27.179	23.715	24.774
TeraGen (M)	21.608	20.567	21.415	20.424	20.352	16.284	20.949	22.223	23.309	19.385
TeraGen (S)	35.781	31.252	27.395	24.367	20.28	28.229	22.226	19.686	18.175	20.482
Grep (L)	407.832	401.472	403.281	400.613	399.036	400.798	402.331	397.04	401.502	403.184
Grep (M)	893.42	891.797	886.995	892.567	888.924	887.631	885.837	886.924	888.883	887.595
Grep (S)	392.327	389.899	389.218	388.905	388.616	388.863	390.007	387.984	390.291	376.054

### 5.3 Virtual Cluster Runs

In this section, we will cover how the jobs performed when moved to a virtual environment in the form of the TCloud cluster presented in 4. The values from the runs are presented in table 5.3. All of the times are measured in seconds. The job presents ten runs on each of the clusters types for all of the different jobs we discuss.

From the results we can see which type of virtual machine clusters was fastest for the different job types. For PI, the Small virtual machine type did end up being the fastest cluster type. For the Pentomino job type, again the Small virtual machine type proved to be the fastest. For Word Count, the Small virtual machine type cluster proved again to be the fastest.

TeraSort also ran the fastest on the Small virtual machine type cluster. For TeraGen, the fastest virtual machine type cluster ended up being the Medium size

cluster. Grep like most of the job types we tested, ended up running fastest on the Small virtual machine type cluster.

In the next chapter we will cover what this means for the two prediction methods presented in chapter 3. We will also cover what these results mean for our two different approaches.

In the next chapter, when we reference the results from the virtual cluster runs on TCloud, we will give the average for each of these jobs on the given type of virtual machine cluster type.

## CHAPTER 6

# DISCUSSION OF RESULTS

### 6.1 Introduction

In this chapter, we will discuss what these results mean for our two prediction methods presented in chapter 3. In subsection 6.2 we will cover the what it means for the Ratio Based Selection. Then in subsection 6.3 we will cover what these results mean for the Best Fit Based selection process.

### 6.2 Ratio Based Selection

Remember previously, we found that TeraGen had a ratio of 9125.2584, Word Count had a ratio of 117.8997, PI had a ratio of 0.1109, DFSIOTest Read had a ratio of 0.0464, Pentomino had a ratio of 0.0194, Grep had 0.0148 as a ratio, MRBench's ratio was 0.0089, DFSIOTest Write had a ratio of 0.0380, and TeraSort had a ratio of 0

This Meant that TeraGen and Word Count should both prefer Larger type machines, especially TeraGen. The other jobs consisting of PI, DFSIOTest Read, Pentomino, Grep, MRBench, DFSIOTest Write, and TeraSort, all should prefer Smaller Machine types.

In reality, this is not how it worked out when the jobs where run on the different virtual machines type clusters. What we found was that only TeraGen preferred the Larger machine type cluster. All of the machines we predicted to prefer the smaller machine type cluster did end up working the best on the smaller machine type cluster.

This means that only Word count was mis predicted. Looking at TeraGen and WordCount, there is a very larger difference in their respective ratio's. This difference ends up being a difference of almost 100 times the size. This could indicate that there may need to be a value similar to the 10 added to the dominator in the other proposed method to help offset some of the other costs not taken into account.

This might be something that with more experience and more work loads, a user will be able to determine. For the most part the predictor did however predict not only the correct type of machine, but also which machines would benefit the most from a certain machine type cluster.

This too, implies that the predictor could possibly be improved by using some constant multiplier based on physical test cluster of the users. How to determine the value of this constant is beyond the topic of this paper.

### **6.3 Best Fit Based Selection**

Recall that PI preferred small machines with a value of 0.8202972, Pentomino preferred a small type of virtual machines, with a minimal value of 0.8146544, Word Count had a minimal value of 0.8079288 for the small type machines, TeraSort preferred a small type of virtual machines, with a minimal value of 0.795928507, TeraGen's minimal positive value was 0.333333322 for the medium type machines, and lastly Grep preferred small type machines with a minimal value of 0.1091744.

When we ran the jobs on the three different virtual machine cluster types, we found that all of the jobs were predicted correctly. This is encouraging and shows a better performance than the Ratio Based Selection discussed in the previous section. There are also some other things of note.

Not only did the predictor correctly predict what type of machines would be best, it also got fairly close to sorting the jobs in terms of improvement between different machine types. This means that a smaller value will get more benefit from a certain

machine type than a job with a higher value. While not perfect, it is encouraging.

By the prediction mechanisms estimation, the Word Count job should have been the next closest, followed by Pentomino and then PI. It was not when we ran the tests on real machines; the order of closeness was Pentomino, PI, and then Word Count.

The only job that was out of order was Word Count. This makes sense, since all three of these jobs have very similar values for both real runs and the prediction mechanisms. Then, in both real runs and the prediction mechanisms, the Terasort jobs were those farthest from needing a larger machine.

Looking closer it gets more encouraging. the difference between the values by the prediction was very similar with the exception of Word Count, to the difference in time in the real runs done on the virtual clusters of different size virtual machines within the TCloud cloud computing cluster.

In the future, we may want to look at how to better determine the constant value used to divide the equation. This could be somewhat complicated, however, after a user determines the constant value for their own cluster, it would not need to be changed. This value should change for each cluster.

What we see over all is very good for both prediction mechanisms. There are some interesting points that can be made and looked at in the future. The first point is that for some reason Word Count give both of the prediction mechanisms problems.

In the Ratio Based Selection, Word Count ended up as the only job that was not predicted for the correct type of virtual cluster within TCloud. Even the Best Fit Based Selection approach which correctly predicted Word Counts type of virtual machine cluster, did not correctly rank it within the jobs by benefit.

While this can be seen as a flaw, it does give us of an idea of where to look to improve both approaches. In the future we hope to look more into why Word Count which is often considered the most basic example job for Map Reduce ended up give both approaches such trouble, and using this knowledge to make prediction better.

We did find that in almost every case, the Best Fit Based Selection did out perform the Ratio Based Selection. With that being said, the Best Fit Based Selection does require more work to get the prediction values. It is possible that a case could arise such as a system with limited computing power which could require the similar approach.

For this, we can see that Ratio Based Selection should be able to still give a very good estimate as to which machine to use. In the future, we may be able to test more jobs and expand our test group of jobs so we better understand how these two approaches work in practice.

## CHAPTER 7

# CONCLUSION

In this paper, we looked at the issue of selecting virtual machines to best fit a MapReduce job. We look specifically at only predicting for one type of job. We assume this to be reasonable since cloud resources are rented by the hour and large MapReduce jobs tend to run for a long time.

In chapter 1, we introduced the problem we worked on solving. We covered the basic structure of MapReduce. We covered the two parts of, Map and Reduce, and we covered how they work together.

We presented a simple example of how this plays out in a word count scenario. It is clear from the example that everyone unique word has been counted correctly. We then presented Hadoop. This means listing who created it and what is meant to do.

Then we cover some of the important specific points of Hadoop including chunk size, and the different phases that exist. We also discuss how these phases work together. The next thing that is covered is the many different schedulers that are offered within Hadoop or by other research papers.

This includes the Fair Share Scheduler, the Capacity Scheduler, and the FIFO scheduler which are all built in to Hadoop. The Dynamic Proportional share scheduler, the Multi-user scheduler, the iShuffle system, a load balancing scheduler, some schedulers for heterogeneous systems all are schedulers presented by other researcher.

We then covered Amazon EC2. We cover how the basic setup of the Amazon EC2. We cover how you use the Amazon and some of the difficulties with using it. This is not the system we use, however it is the system comparable to the system we

use.

We then covered Eucalyptus. This software is presented as a comparable software to Amazon EC2. We covered the intricacies of setting up the Eucalyptus software. We also cover how it was recently purchased and now owned by HP.

Next in chapter 2, we did a review of the literature around our issue. We covered some different implementations of Map Reduce. Then we cover issues related to our own paper. This includes papers that work on Hadoop speed up, papers that deal with similar issues, and papers that deal with MapReduce in the Cloud.

The Implementations we covered included Hadoop, CouchDB, Infinispan, MongoDB, and Riak. When we cover Hadoop, we also discuss some of the other Apache projects that are related to or add onto Hadoop.

With the exception of Hadoop, the other softwares that are presented only use the concept of MapReduce within themselves. Most of the softwares are data bases that use MapReduce to do some of their processing.

The next part that we cover is papers that relate to our own work presented here. Some of the papers that we cover are works relate to how others have attempted to speed up Hadoop.

This includes the papers titled, Dynamic Proportional Share Scheduling in Hadoop, Job scheduling for Multi-user MapReduce Clusters, iShuffle: Improving Hadoop Performance with shuffle-on-write, Online Load Balancing for MapReduce with Skewed Data inputs, Tarazu: Optimizing MapReduce on Heterogeneous Clusters, and Improving MapReduce Performance in Heterogeneous Environments.

The papers that are presented after that, attempt to solve a problem that similar, but not exactly the same as our own. These papers include, Exploiting Cloud Heterogeneity for Optimized Cost Performance MapReduce Processing, and Optimizing Cost and Performance Trade-Offs for MapReduce Job Processing in the Cloud.

What makes the problems presented in theses papers different from our own, is

they assume that a user has a workload consisting of multiple types of MapReduce jobs. In our paper we make the assumption that we are only trying to schedule for one type of MapReduce job at a time.

The papers presented after these cover attempts to run MapReduce in the cloud. These papers include, Evaluating MapReduce on virtual Machines: the Hadoop Case, A Performance Study on the VM Startup Time in the Cloud, CLOUDLET: towards MapReduce Implementation on virtual machines, Scaling and Scheduling to Maximize Application Performance Within Budget Constraints in Cloud Workflows, and Towards Optimizing Hadoop Provisioning in the Cloud.

After that in chapter 3 we cover our two approaches to selection. I discuss the first approach, Ratio Based selection, and how it works. From there I cover the Best Fit Based selection and it is implemented.

With the Ratio Based Selection, we cover some of the parts of MapReduce that lead us intuitively to this approach. This means looking both the basic MapReduce paradigm as well as the framework and inner workings of Hadoop.

We also cover some of the different metrics that are available to us about a Hadoop job. We cover what these metrics mean from a systems stance. We also cover how we might be able to or not able to use these metrics

We then offer up our first attempt at using these metrics for prediction of virtual machine type. We find that it does not work. From there we move to the method that we did find to work and we go into detail of how to use it.

After we present how this method works for prediction, we cover how it predicts the different jobs that we test with. These values that we present are used later to show the effectiveness of the algorithm.

After this we move on to present our other method of selection call the Best Fit Based Selection. Again we cover why this approach makes sense intuitively based on the MapReduce paradigm and the framework given by Hadoop.

We again cover the different metrics that are available to use. We cover what they mean, and how they might be used to predict what type of virtual machine to use. This is different from the last approach because we use them differently.

In this section we then move to which specific metrics we decide to use and why we decided to use them. We then give names to different metrics that will be used later for prediction of the best virtual machine type for a MapReduce job.

After this we move on to how the actual prediction is done. This is somewhat more complicated than the other approach and we go more in depth with the different parts that are used for prediction.

In chapter 4, I cover the different systems that we use. We cover what they system are as well as covering how we used them throughout the thesis. We also covered the setup of some different parts of the systems.

The first system covered in this section is the Net Cloud computing cluster. This cluster is a cluster that our group built and setup for the purposes of testing. In this paper we cover many different aspects of this cluster.

One of the things we cover about Net Cloud is the physical makeup of the different parts of the cluster. This includes the specifications and makes and modes for the different hardware used within the cluster.

Another thing that we cover about Net Cloud is the different softwares that are used in the cluster. This includes the operating systems, what version of Hadoop and other softwares that could be used.

The final things cover about the Net Cloud cluster is the topology of the cluster. We go in depth about the topology and also cover why and how the topology was put into place.

The second system that we cover in this section is the TCloud computing cluster. This is a cluster which is setup for general cloud computing research within Temple's computer science department.

We cover the as much as possible the physical makeup of the computing cluster. While we do not know some specifics of the different servers, we are able to information about the total number of servers and some general information about the power of all of them.

Next we cover how the cluster is used within our research. This means explaining what type of software in install that we take advantage of. This again is someone tricky as we do not know some of the specifics.

In this section we also cover the different types of virtual cluster that we have available to use. These cluster within TCloud are make up completely of virtual machines located throughout TCloud.

To wrap the chapter up we cover the topology of TCloud. This is any interesting topology for a few different reasons. One reason is the power and speed that is offered. Another reason it is interesting is due to the fact that it does not give us how our virtual machines will end up communicating since they are virtual.

Then, in chapter 5, we cover the results from doing experimentation. This means testing on our two different types of clusters. These results are then presented and covered in how they relate to the two different prediction mechanisms.

They physical runs take place on the Net Cloud Cluster. The first thing that is presented is the raw data. This means giving the average values for each of the different metrics that where covered earlier in the paper.

The next thing that is covered is what these values mean for the Ratio Based Selection mechanism. This means determining the values for the different jobs and say what theses different values mean for selection.

After this we cover the Best Fit Selection mechanism. Again this means determining the values for each of the jobs. This is a little more complicated as there are different parts to calculate for this approach. Then again we move on to giving the prediction results based on theses values.

From here we move on to covering the results from the three different types of virtual cluster that are hosted on TCloud. These results are purely times that it takes the different jobs on different types of virtual machine clusters to run to completion.

Finally, in chapter 6, discuss what the results presented in chapter 5 and what they mean for the two different prediction mechanisms presented in chapter 3.

For Ratio Based Selection, we repeat what the different values are for the different jobs that are run within the physical cluster. We then look at the results from the three virtual clusters that are run on TCloud. We discuss not only whether the values are correct, but if it correctly predicted which jobs would benefit most from the selected machines.

After this we move on to the Best Fit Based Selection. We again present the value for the different jobs from the previous chapter. Again we compare how the predictions hold up to the results from three different virtual clusters run on TCloud. Just like before we look not only at correctness but also at if the benefit has been correctly predicted.

For both of the prediction mechanisms we also cover the future need for a constant scaling value. We cover that this value would have to be determined independently based on each users physical test system.

That brings us to this chapter, chapter 7 the Conclusion. By now we thoroughly presented a review of the literature around this problem. We have presented our two different mechanisms for prediction of virtual machine type for building a cluster based on a MapReduce job.

We have gone into why these two mechanisms make sense intuitively. We have presented how they work in practice and discussed what the results from testing mean for the two prediction mechanisms.

Furthermore we have presented our experiences why building and working with our two different types of systems. This should allow readers to easily implement

systems similar to those that we use for the purposes of this paper.

We have also covered some of the places we would like to go with future work into this subject matter and problem. This includes, finding constant multiplying values, and looking into what makes the Word Count job such a problem for our prediction mechanisms.

## BIBLIOGRAPHY

- ( ), Temple's tcloud, <https://sites.google.com/a/temple.edu/tcloud/>.
- (2008), Apaches's hadoop capacity scheduler, [http://hadoop.apache.org/docs/r1.2.1/capacity\\\_scheduler.html](http://hadoop.apache.org/docs/r1.2.1/capacity\_scheduler.html).
- (2008), Apahce's hadoop fair scheduler, [http://hadoop.apache.org/docs/r1.2.1/fair\\\_scheduler.html](http://hadoop.apache.org/docs/r1.2.1/fair\_scheduler.html).
- (2014), [http://www.ibm.com/ibm/puresystems/us/en/pd\\_hadoop.html](http://www.ibm.com/ibm/puresystems/us/en/pd_hadoop.html).
- Ahmad, F., S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar (2012), Tarazu: optimizing mapreduce on heterogeneous clusters, in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 61–74, ACM.
- Amazon, E. (2010), Amazon elastic compute cloud (amazon ec2), *Amazon Elastic Compute Cloud (Amazon EC2)*.
- Bialecki, A., M. Cafarella, D. Cutting, and O. OMalley (2005), Hadoop: a framework for running applications on large clusters built of commodity hardware, 2005, *Wiki at http://lucene.apache.org/hadoop*.
- Blaisse, A. P., M. Berlove, and J. Wu (2014), Setup and configuration of mapreduce in a cloud environment, in *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pp. 773–774, IEEE.
- Borthakur, D., et al. (2011), Apache hadoop goes realtime at facebook, in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 1071–1080, ACM.
- Ghemawat, S., and J. Dean (2004), Mapreduce: simplified data processing on large clusters, in *Proc. OSDI*.
- Guo, Y., J. Rao, and X. Zhou (2013), ishuffle: Improving hadoop performance with shuffle-on-write., in *ICAC*, pp. 107–117.
- Hadoop, A. (2011), Apache hadoop.
- Ibrahim, S., H. Jin, B. Cheng, H. Cao, S. Wu, and L. Qi (2009a), Cloudlet: towards mapreduce implementation on virtual machines, in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pp. 65–66, ACM.

- Ibrahim, S., H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi (2009b), Evaluating mapreduce on virtual machines: The hadoop case, in *Cloud Computing*, pp. 519–528, Springer.
- Kambatla, K., A. Pathak, and H. Pucha (2009), Towards optimizing hadoop provisioning in the cloud.
- Le, Y., J. Liu, F. Ergün, and D. Wang (), Online load balancing for mapreduce with skewed data input.
- Mao, M., and M. Humphrey (2012), A performance study on the vm startup time in the cloud, in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 423–430, IEEE.
- Mao, M., and M. Humphrey (2013), Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 67–78, IEEE.
- OMalley, O. (2008), Terabyte sort on apache hadoop, *Yahoo*, available online at: <http://sortbenchmark.org/Yahoo-Hadoop.pdf>, (May), pp. 1–3.
- Sandholm, T., and K. Lai (2010), Dynamic proportional share scheduling in hadoop, in *Job scheduling strategies for parallel processing*, pp. 110–131, Springer.
- Weil, K. (2010), Hadoop at twitter, *Twitter Engineering Blog*, 8.
- Zaharia, M., A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica (2008), Improving mapreduce performance in heterogeneous environments., in *OSDI*, vol. 8, p. 7.
- Zaharia, M., D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica (2009), Job scheduling for multi-user mapreduce clusters, *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*.
- Zhang, Z., L. Cherkasova, and B. T. Loo (2014a), Exploiting cloud heterogeneity for optimized cost/performance mapreduce processing, in *Proceedings of the Fourth International Workshop on Cloud Data and Platforms*, p. 1, ACM.
- Zhang, Z., L. Cherkasova, and B. T. Loo (2014b), Optimizing cost and performance trade-offs for mapreduce job processing in the cloud, in *Proc. of IEEE/IFIP NOMS*.
- Zikopoulos, P., K. Parasuraman, T. Deutsch, J. Giles, D. Corrigan, et al. (2012), *Harness the Power of Big Data The IBM Big Data Platform*, McGraw Hill Professional.