1-1-2012

Systems support for genomics computing in cloud environments

Tung Thanh Nguyen
Wayne State University,

SYSTEMS SUPPORT FOR GENOMICS COMPUTING IN CLOUD ENVIRONMENTS

by

TUNG NGUYEN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2012

MAJOR: COMPUTER SCIENCE

Approved by:

____

Advisor Date

____

____

____

DEDICATION

To my beloved family.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratefulness to my advisor, Prof. Weisong Shi, for his invaluable and continuous support during my Ph.D. program. His research enthusiasm has inspired me to pursue the degree. During my study at Wayne State University, he has patiently spent numerous hours with me discussing research ideas, guiding me to write papers, and giving me comments. Moreover, when I lose my confident on my work, he always encourage me to keep on. Not only helping us on research and education, he also pay attention to our daily life difficulty. He always available to me and other students whenever we needed his feedback.

I am also very grateful to Prof. Brockmeyer, Prof. Fisher, and Prof. Jiang for serving on my dissertation committee and giving constructive suggestions on improving the dissertation. I would like to thank Prof. Douglas Ruden, Director of Epigenomics lab in the Institute of Environmental Health Sciences, for his generous support to my work related to genomic research.

Also, I thank all my colleagues in the MIST Lab, LAST group and the Epigenomics lab (Prof. Ruden), my friends Dr. Guoxing Zhang, Shinan, Masud, Pablo, Dr. Kruger, and many more for having insightful discussions with me.

Last but not least, I deeply acknowledge the love and encouragement from my wife (BAO TRIEU), my father (DON NGUYEN) and my mother (MAI NGUYEN). I hope I have made them all proud.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

In this chapter, we first introduce the motivation then elaborate the objectives as well as the contributions of our work. Before going into the detail, we will outline the rest of the thesis in the end of this chapter.

1.1 Motivation

1.1.1 Application pull

New sequencing technologies in genomics, like other sciences, create enormous amounts of data to process. Manufacturers are constantly increasing output in terms of a number of reads, increasing read length, as well as working to improve read quality. While it took 10 years and \$3 billion dollars to produce a first draft of the human reference genome [58] (approx. 3.5 billion base pairs), the current generation of sequencing instruments is able to generate hundreds of billions of bases in only a few days and this output will continue to increase dramatically over the next few years. For example, the latest sequencer from Illumina, the HiSeq 2000, is able to generate 25 G bases of sequence per DAY [118]. In term of price, in comparison to the GA sequencer, the cost per base is actually substantially reduced on the HiSeq by as much as 8 times [100]. Recently, Illumina has announced their newest improvement on HiSeq 2000 sequencer that will almost triple its output - to 600 gigabases per run-while reducing reagent costs for sequencing a human genome at least two-fold [78]. However, this is still the second-generation sequencing technology. The third-generation single-molecule sequencing instruments, which are beginning to be introduced, by Pacific or Helicos Biosciences promise to reduce the output cost even further.

In the race between DNA sequencing throughput and computer speed, remarkably, sequencing is winning by a mile. While sequencing throughput has increased at a rate of about

5-fold per year, computer speed generally follows ”Moore’s Law,” doubling every 18 or 24 months. As this gap widens, there are many serious problems and challenges towards managing these extensive genetic informational datasets. The first challenge is performance. As the data grows it is taking an increasing amount of time to compile, search and analyze and radical new approaches are required that would ensure project scalability. The second issue is the enormous capital expense for equipment that typically has a 6-month as state-of-the-art half-life. A greater challenge is data redundancy. Both the Illumina GAII and AB SOLiD systems were developed primarily for re-sequencing genomes used to identify variation between individuals (single nucleotide polymorphisms [SNPs]and insertions/deletions [InDels]) [30]. The vast majority of data produced by these re-sequencing experiments will be identical to the reference sequence. This redundancy issue will increase to a point where storage within the primary data repositories becomes impractical.

1.1.2 Technology push

In computer science, Cloud Computing has recently emerged as an evolutionary model to accommodate storage and computing service as a utility. Cloud providers offer different computing services to users through the Internet. Cloud users only pay for the resources (computing, bandwidth, etc) they actually consume without worrying about the maintenance expense, provisioning resources for future needs, taking care of availability, and reliability issues. The price is based on the time and types of services. As a result using Cloud Computing services is a recent and very promising solution in bioinformatics to deal with the issues related to storage and computation.

The Cloud Computing solution, however, just enables the flexible and scalable infrastructure to deal with storage and computation issues. To deal with performance and scalability when processing a huge amount of data, we need to have a special parallel programming model. Recently, Google has designed a parallel computing framework called MapReduce [49] which can scale efficiently many thousands of commodity machines. These commodity machines forming a cluster can be accessed by users in an institution or can be rented over the Internet through utility computing services. Actually, the idea of this framework is not new since it has already been used in traditional functional programming languages such as Haskell, Lisp, Erlang, etc.

In general, the combination of the elasticity of the Cloud and the scalability of MapReduce is a very promising solution for Genomic challenges.

1.2 Objectives

After realizing the problems that bioinformaticians are facing in Genomics, as well as the chance to address them by applying Cloud technologies, in this work, we take the first step to connect the needs (in Genomics) and the supply (in Compute Science). In other words, we will apply Cloud Computing and MapReduce to manage and process the Genomic data. Specifically, the objective of this research is to create an innovative application specific data processing and management system for *large genomic datasets* in the Cloud. This new genomic data management and processing technique enables genomic researchers' access to data with reduced cost and maintenance, and faster mining capabilities at an unprecedented level.

The particular aims of this dissertation include:

1. To study the advantages (or disadvantages) of applying Cloud Computing in Genomic research.

2. To study how heterogeneity affects the performance/effectiveness of the system and how to tame it.

3. To analyze the storage and processing requirements of particular bioinformatics projects including data formats, data access pattern, security and business models.

4. To reduce/minimizing the data transmission when executing analysis jobs inside the cloud. As pointed earlier in this section, the current MapReduce in the cloud model such as the Amazon EMR wastes a lot of resource on data movement inside the data

center. The expected output would be a system that is able to reduce the traffic of data between computing servers and storage servers in the cloud.

5. To analyze, model, design and implement a simple yet powerful tool allowing scientists to upload data and run analysis easily and securely in the Cloud with minimal respond time. If the previous target is about the internal data movement, this one aims not only at the external data movement but the computing time also. Together they provide a complete solution to deal with data movement and processing overhead in the considering system. The result of this goal is an easy-to-use software tool that is used by the domain experts to submit jobs and (big) data to the system. The tool will take care of data transferring, processing (with MapReduce framework) and sending the results back with the smallest respond time.

6. To study how to improve the performance, energy consumption and effectiveness of the system.

1.3 Summary of Contributions

The main contributions are listed as the followings:

1. We developed CloudAligner - a fast and full-featured MapReduce based tool to align sequences onto chromosomes. The performance gain of CloudAligner over Cloud-based counterparts is from 35% to 80%. We also converted an popular application in metagenomics to MapReduce style to improve it scalability as well as to serve as a real-life application for our SPBD (Streamlining Processing for Big Data) system.

2. In terms of system, we developed tools such as OPERA and DiR to tame the heterogeneity of the underlying infrastructure and enable differentiated services. The side effects of these are the improvement in resource usage and performance.

3. We also introduced a new metric to capture the usage efficiency of a system. We showed

that while the efficiency computed from the traditional method is 100%, it is less than 44% with our new metric. This suggests that the current metrics are inadequate to capture the efficiency of a system.

4. We also developed a distributed cache system to minimize the overhead of cloud internal data movement. Under the warm cache scenario, which is usually true once the cloud starts running for a while, our system can achieve performance improvement by up to 56.4% with two MapReduce-based applications in lifesciences (genomics), 75.1% with the traditional Sort application and 83.7% with the Grep application.

5. A new tool (SPBD) was also developed to minimize the respond time of the system while analyzing big data stored outside the cloud. The results shows that SPBD can improve the respond time of wordcount and metagenomic application 34% and 31% with 32GB of input data, respectively.

1.4 Our Approach

To ensure the sustainability and usefulness of our research, we use application demand/problem and modern infrastructure characteristics/trends to drive our design. As the authors are Computer Scientists, who do not have much knowledge in bioinformatics, the first step would be to cooperate with domain experts (bioinformaticians and genomics scientists) to have a better understanding of their challenges in terms of data management and computing. From there, we can derive (in the computer system point of view) the requirements as well as challenges of our big data management system and address each of them accordingly. Finally, we will consider some possible refinements (efficiency) for our system. We plan to attack the problem from different layers ranging from application to system and data movement. Figure 1.1 shows the overview of the system with different layers and our work on each of them.

Following the approach, the next chapter will review the general background as well as related work in both Genomics and Computer Science. Then in chapter 3, we will show that

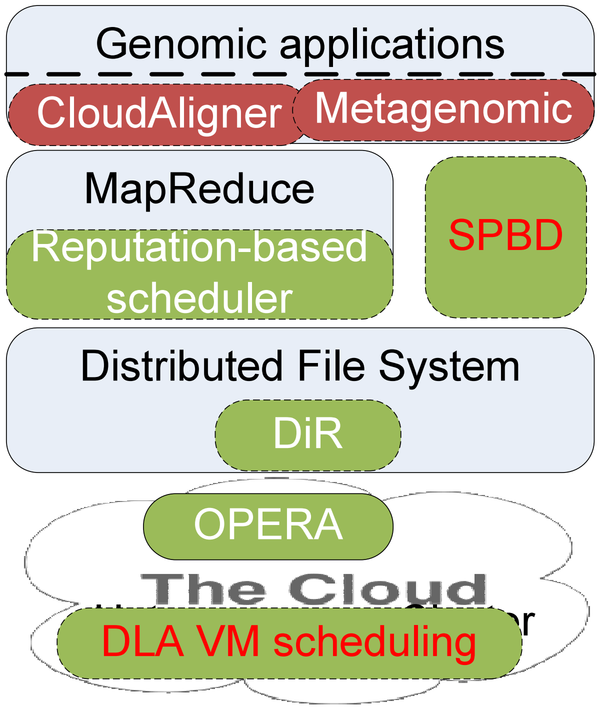


Figure 1.1: System Overview.

using Cloud Computing is a suitable solution for processing Genomic data by developing a new MapReduce-based application for sequence mapping called CloudAligner which outperforms its on-premise and even MapReduce-based counterparts. Next, Chapter 4 and 5 will discuss about the issues related to the heterogeneity and detail our proposed solutions for each of them. After that, Chapter 6 and 7 are presented to propose tool and solutions to handle big data movement in the system. Finally, following chapter8 that discusses about the efficiency of the system, chapter 9 gives the conclusions and future work.

For clarity, at the beginning of each chapter (from Chapter 3 to 7), we will have a figure summarizing what parts of the whole system that chapter is targeting.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter contains two sub sections: 1Genomic related background and 2Virtualization and Cloud Computing. The first one sketches the related background in genomic research. Its intent is to help readers in computer scientists can have better understanding about the content of the thesis. The second sub section serves as a summary of the techniques or terminologies in Cloud Computing area. With such brief background information, we hope that the readers can have a better understanding of our work.

2.1 Genomic related background

2.1.1 Basic definitions

In this sub section, the definitions of some basic genomic terminologies will be provided as references to help pure Computer Scientists understanding our work more easily. Most definitions here come from wikipedia [6]). It's worth to note, however, that not all definitions are referred. Some important but fundamental definitions/concepts such as nucleus, cell, protein, chromosome, DNA, RNA, etc will not be mentioned.

Base: Nucleobases (or nucleotide bases/nitrogenous bases/aglycones) are the parts of DNA and RNA that may be involved in pairing. The primary nucleobases are cytosine, guanine, adenine (DNA and RNA), thymine (DNA) and uracil (RNA), abbreviated as C, G, A, T, and U, respectively. They are usually simply called bases in genetics. Because A, G, C, and T appear in the DNA, these molecules are called DNA-bases; A, G, C, and U are called RNA-bases.

Base pair: In molecular biology and genetics, two nucleotides on opposite complementary DNA or RNA strands that are connected via hydrogen bonds are called a base pair (often abbreviated bp). In the canonical Watson-Crick DNA base pairing, adenine (A) forms a base pair with thymine (T) and guanine (G) forms a base pair with cytosine (C).

Table 2.1: The sequencing technology.

Machines	**Sequencing technology**
Roche 454 GS FLX	Pyrosequencing
Illumina SOLEXA	Sequencing by synthesis
Applied Biosystems	Sequencing by ligation
SOLID VisiGen Biotechnologies	Single-molecule sequencing
Helicos BioSciences	Nanopore sequencing

Gene: a unit of heredity in a living organism.

Genetics: a discipline of biology, is the science of genes, heredity, and variation in living organisms

Genome: in modern molecular biology and genetics, the genome is the entirety of an organism's hereditary information. It is encoded either in DNA or, for many types of virus, in RNA. The genome includes both the genes and the non-coding sequences of the DNA/RNA.

Genomics: a discipline in genetics concerning the study of the genomes of organisms.

Sequencing: in genetics and biochemistry, sequencing means to determine the primary structure (or primary sequence) of an unbranched biopolymer. Sequencing results in a symbolic linear depiction known as a sequence which succinctly summarizes much of the atomic-level structure of the sequenced molecule.

In this work, we focus on sequencing in genomics because it is where a vast amount of data is generated.

2.1.2 How sequence data is generated?

Normally, sequence data is generated from the sequencer. Table 2.1.2 lists 4 major new sequencing technologies together with 4 machines applying them, and Table 2.1.2 compares the 3 most popular machines in term of sequencing method, base per run, read length, execution time and cost [134].

Up to now, there have been 3 generations of sequencers. The definitions of them are not quite clear. Based on [76], the first generation sequencing technology changed the way we thought about DNA ”from something that we could glimpse to something we could sequence

Table 2.2: A comparison between sequencers.

	Roche/454	ABI Solid	Illumina SOLEXA
Sequencing method	Pyrosequencing chemistry	Oligo ligation cleavage	Labeled base addition
Base per run	100 million	1000-1500 million	1000-3000 million
Read length	100-200 bases	35 bases	35-50 bases
Length of run	7.5 hrs.	2-4 days	2-3 days
Cost of just sequencing	$9,000 per run	$5,000 per run	$3,500 per run

on mass." The second generation sequencing has changed it "from something we sequence once, to something we sequence again and again." The majority of sequencers using today are of the second generation. The third generation have just introduced recently. The so-called third generation sequencers are expected to sequence single molecules of DNA in real-time and overcome some constraints from the second generation sequencing such as low read lengths, long execution time, using expensive enzymes. The pictures of some real second and third generation machines are shown respectively in Figure 2.1 and 2.2.

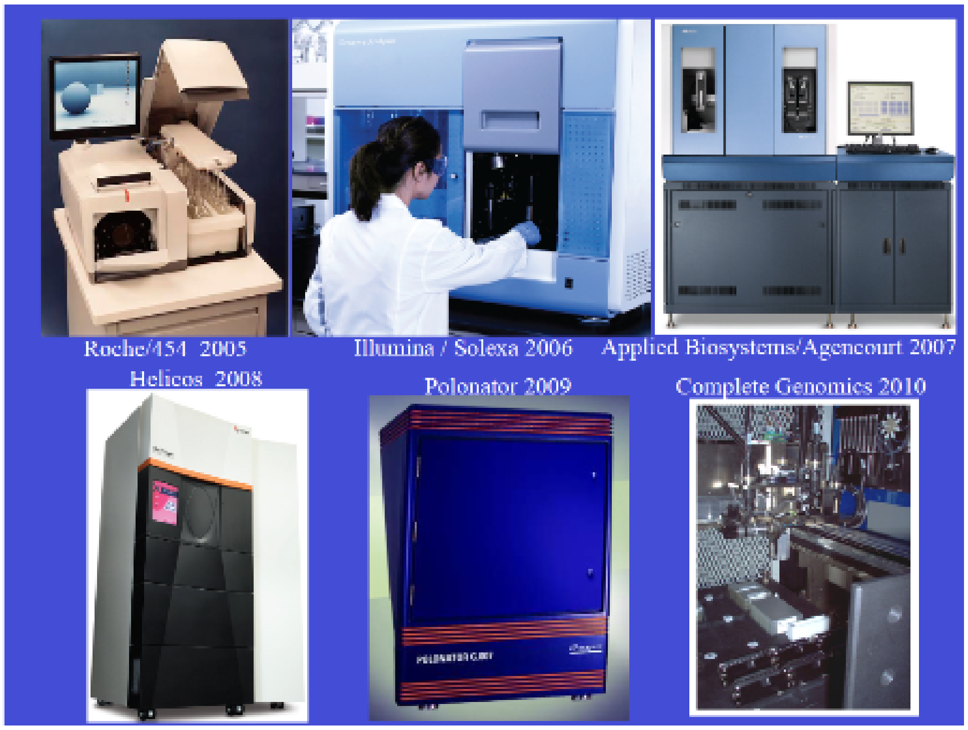


Figure 2.1: Second Generation DNA Sequencing Technologies [1].

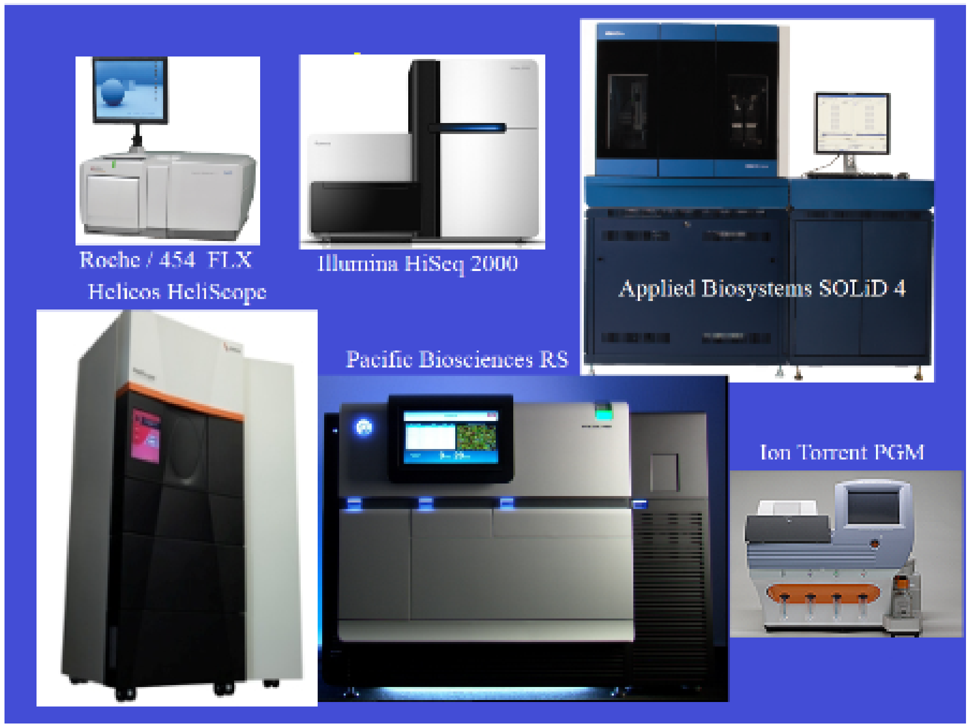


Figure 2.2: Third Generation DNA Sequencing Technologies [1].

The data directly generated from the sequencers cannot be used immediately in biological analysis. It has to go through a typical sequencing pipeline as shown in Figure 2.3. From the figure, the raw data output from the machines are just images (level 0 data). Although occupied a large portion of storage devices, for the sake of carefulness, most of these data were still not be discarded. However, currently they are deleted by default in some modern machines. As a result, they will not be conceived in our study.

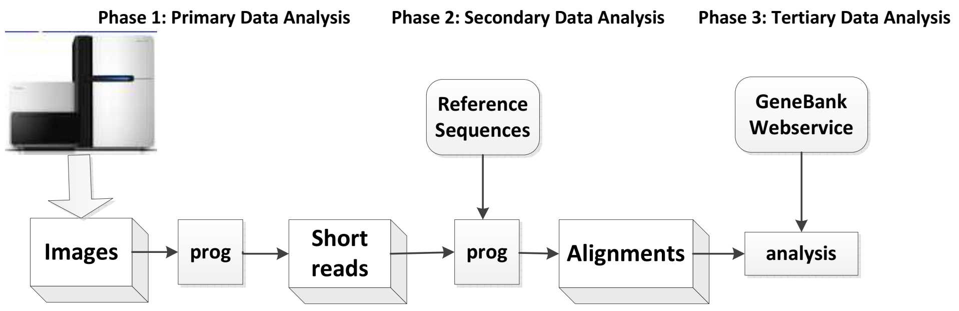

Phase 1: Primary Data Analysis
Phase 2: Secondary Data Analysis
Phase 3: Tertiary Data Analysis
Reference Sequences
GeneBank Webservice
Images
prog
Short reads
prog
Alignments
analysis

Figure 2.3: Typical Gene Sequencing Pipeline [114].

The level 1 data in Figure 2.3 are short reads (nucleotide sequences). They are actually just text files. There are many formats for these files including fasta, fastq, CSFASTA, qseq, sff, FNA/QUAL, etc.

These short read files are then mapped by a software like CloudAligner (see Chapter 3) to the reference genomes/sequences to produce alignment files. These files are also text file. They are usually in many formats such as SAM, BAM, BED, etc.

These alignments are then looked up into the gen bank to identify which genes included in the reads. From here, the biologist and bioinformatician can start there specific analysis.

2.2 Virtualization and Cloud Computing

Virtualization is the core technology that Cloud Computing is based on. Therefore, in this section, we will first study the virtualization then Cloud Computing.

2.2.1 Virtualization

Virtualization has recently been a hot topic, and this chapter is going to present a general overview of it. In this chapter, the definition the virtualization and related concepts will be given first. Then, we will go over its history. After that, we will examine the motivation of virtualization as well as its abilities and benefits. Different types of virtualization, some main players as well as products in the field are introduced next. As specific examples, two case studies will also be detailed. The two final sections point out certain issues of virtualization and reading lists where interested readers can find more information about virtualization.

Definitions

In this section, we are going to define what is virtualization and related terminologies such as virtual machine, virtual machine monitor, hypervisor, host/guest Operating System (OS), etc.

There are many different definitions of Virtualization on the Internet. Many of them are essentially too specific or focus on the Virtual machine definition only. For example, the definition “virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others.” [124] is too specific. The term “virtualization” does not imply partitioning only. For example, Grid computing basically provides the “virtualization” of distributed resources such as storage, bandwidth, CPU cycles, and so on. Another example is PVM (Parallel Virtual Machine) which is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer [2].

The definition, which is concise and generally enough, is of wikipedia: “*Virtualization* is a term that refers to the abstraction of computer resources” [6]. The resources that can be abstracted vary ranging from disk blocks, storage devices, OS, network resources, processes to the whole physical machine. There are many kinds of abstractions. For example, disk blocks are abstracted to files; multiple storage devices are abstracted to a logical volume; a logical volume can also be an abstraction of only a partition of the hard drive; the processes are the abstraction of memory and hardware resources; and a computer intermediate language such as Java bytecode is the abstraction of real machine code. Virtualization also means something unreal.

From the general definition of virtualization, we can easily understand the other more specific definitions such as *network virtualization*, *storage virtualization*, *server virtualization*. *Network virtualization* abstracts the network bandwidth by splitting it up into separated channels which can be assigned to particular devices. *Storage virtualization* is the abstraction of physical storage from multiple network storage devices into a single storage device. It is often used in storage area networks (SANs). *Server virtualization* is the abstraction of server resources (physical servers, processors, and operating systems) from server users [6].

"A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine" [6]. As we can see from the definition of virtualization above, the VM is simply one type of virtualization. The VM is the abstraction of a physical machine. However, the VM has recently been so popular that once somebody mention virtualization, it is most likely that he or she is talking about VM. Therefore, from now on, we will focus more on the VM than on other types of virtualization. However, we would try to keep the diversity and generalization of virtualization as much as possible.

A *host OS* is the original operating system installed on a computer. A *guest OS* is an operating system that is installed in a virtual machine or disk partition in addition to the host OS. In virtualization, the guest OS and host OS can run at the same time.

Generically, in order to implement VM, one would use a layer of software to provide the abstraction of the real machine. The software that implements the virtual machines is called the Virtual Machine Monitor (VMM) or hypervisor. It allocates hardware resources dynamically and transparently. The formal requirements of VMM can be found in [107]. There are many VMM architectures [20]:

- Full emulation: the instructions are handed to the emulator where they are converted into a other instruction set to be executed on the real underlying physical machine. It is often used during the development of software for new hardware which might not be available yet.

- Hypervisor based (type 1): the VMM is directly installed on top of the hardware therefore the performance overhead in this approach is very small. The VMM provides virtual hardware abstractions to create and manage multiple VMs. In this case, the VMM can be considered as the minimal OS.

- Hosted (type 2): the VMM is installed as a service in the host OS. It creates and manages multiple virtual machines as processes. Each virtual machine process contains a full guest OS. Although this approach can greatly simplify the design of the VMM, its overhead is costly because it depends on the host OS API to do anything. The VMM does not directly control the hardware.

Why Virtualization?

After knowing what is virtualization, it's time to answer a very interesting but fundamental question: why we need virtualization? Generally, a new technology is developed because it helps solving certain problems or bring many benefits. In this part, we will study what problems that originated the virtualization only.

In the mainframe stage, since mainframes were expensive resources, it was too wasteful to leave them underutilized. As a result, one machine was often shared by many users. However, as the matter of fact, almost all software applications have bugs. If an application was crashed, it may interfere with other users on that machine. For example, it accquires the shared resources and stalls. Another example of the interference is in the deploying/installing stage of new applications. It often require the machine to be rebooted. Therefore, virtualization was developed to share the scare and expensive resources among users more effectively, separately and securely. Later, in the x86 stage, as the cost of hardware has been going down and computer was more powerful (see Figure 2.4), the virtualization was no longer needed. Each person can even own a powerful computer easily.

In the x86 machines, at first, the system designers want to build operating systems to wrap the hardware to ease the application development in the upper layers. As the computer was getting more popular, the computer hardware market became very attractive. A lot of companies emerged to supply the computer hardware devices. They competed with each other. Although this helped to lower the price, it created the incompatible and communication problems between devices. It is this heterogeneity creates the needs for an abstraction layer of hardware devices - Operating Systems (OSes). The popular of Microsoft Windows OSes is an undeniable proof for the success of this idea.

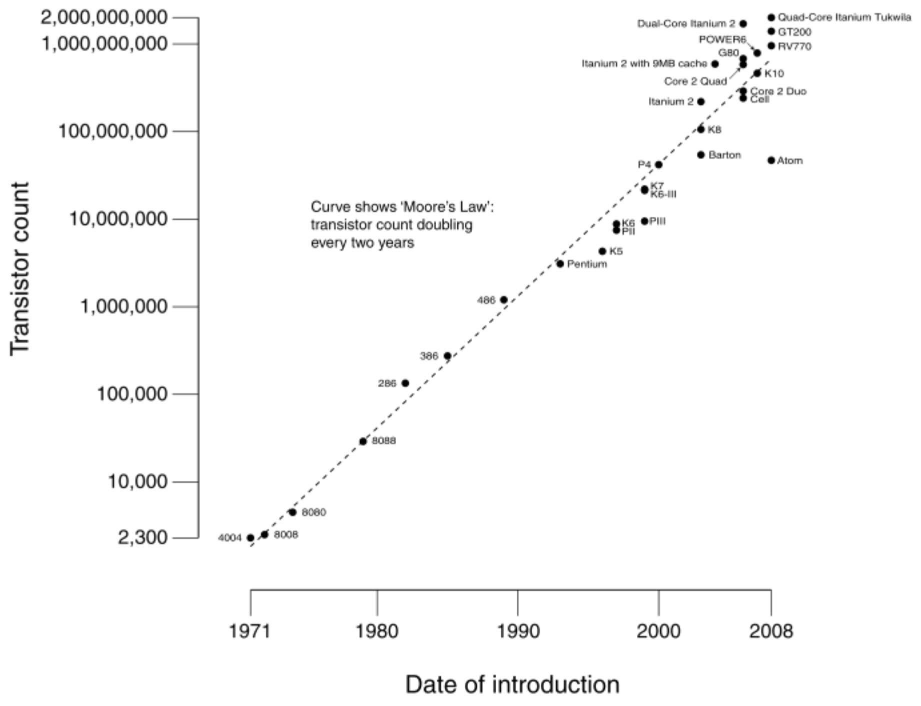


Figure 2.4: The growth of CPU in terms of number of transistors and Moore's Law [6].

It is the features that made the OSes attractive created this obstruction and resulted in the resurrection of virtualization. Robert P. Goldberg said in [61] that "Virtual machine systems were originally developed to correct some of the shortcomings of the typical third generation architectures and multi-programming operating systems." The first shortcomings of single OS approach is that it exposes only one "bare machine interface" which allows only one kernel (OS) to be run at a given time. Second, once the system is running, no disruptive activities such as upgrade, migration, system debugging and so on are allowed. Third, we are not able to run untrusted or newly developed applications in a secure manner (sandbox). Fourth, it is hard to provide the emulation of a newly developed (not available yet) hardware to some software [124]. Fifth, the OSes were closely coupled with the hardware. As the hardware becomes obsolete, so does them. Microsoft, for instance, keeps producing the new OSes to take full advantages of the newly developed hardware.

In addition, as the time goes, the complexity of software, including the OS, was increasing

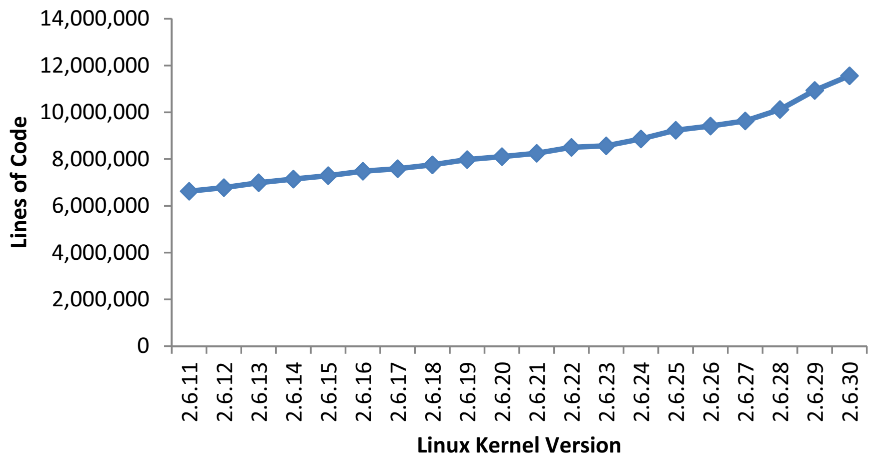


Figure 2.5: Growth of the GNU/Linux kernel [62].

(see Figure 2.5). Each application often requires different hardware and/or platform configurations. Some software was developed for a certain OS only. We couldn't run many such applications on the same physical machine with only one OS. For example, we can't run MS SQL Server 2008 on Windows 2008 Server and, at the same time, run Oracle on Linux on the same hardware.

As a result, many administrators started to apply "one application per server" model for reliability, isolation and security [15]. For example, one might use one server for an Oracle instance, one for a Microsoft instance, yet another for an Exchange Server and so on. However, this approach results into a problem called server sprawl. Server sprawl created too many servers to manage, which consume more power, waste the resources because some (if not many) of the servers are under-utilized. Actually, most server systems are under-utilized as the x86 machines become more powerful (see Figure 2.4). General estimates showed that the average utilization of Windows server was at around 15% and that of UNIX was at 20-30% which was really low [57]. It's even less with dual core and quad core processors. However, the cost to operate a data center, power and cooling, is very high. Consequently, the problem of utilization of resources in the earlier stage (mainframe) has started surfacing again and hence the virtualization returns to the mainstream.

The return of virtualization may be marked by a server virtualization solution for x86 machines in 1999 of VMware. VMs have decoupled the operating systems from hardware by introducing a virtualization layer called virtual machine monitor (VMM) [20]. This introduced a completely new way of looking at OSes.

2.2.2 Cloud Computing

In computer science, Cloud Computing has recently emerged as an evolutionary model to accommodate storage and computing service as a utility. Cloud providers offer different computing services to users through the Internet. Cloud users only pay for the resources (computing, bandwidth, etc) they actually consume without worrying about the maintenance expense, provisioning resources for future needs, taking care of availability, and reliability issues. The price is based on the time and types of services. As a result using Cloud Computing services is a recent and very promising solution in bioinformatics to deal with the issues related to storage and computation.

There are many Cloud service providers including three big companies Amazon, Google, and Microsoft. Amazon has EC2, Microsoft has Windows Azure and Google has App Engine. Although they all provide cloud services, they are different from each other. Amazon EC2 provides virtual machines called instances with which user can do whatever they want as if they are physical machines. They can install any OS, software they need on these instances. With Windows Azure, user do not have the option to use any other OS than Windows. However, since it is developed by Microsoft, it is very easy to develop application using Visual Studio. A developer who is familiar with Visual Studio will find that there is almost no difference in developing software locally and in the Cloud. Finally, Google App Engine provides a framework to develop applications. Specially, they focus more on building and hosting web applications. In this work, we choose to build our tool on Amazon EC2 since as a research problem we want to have more control over the system.

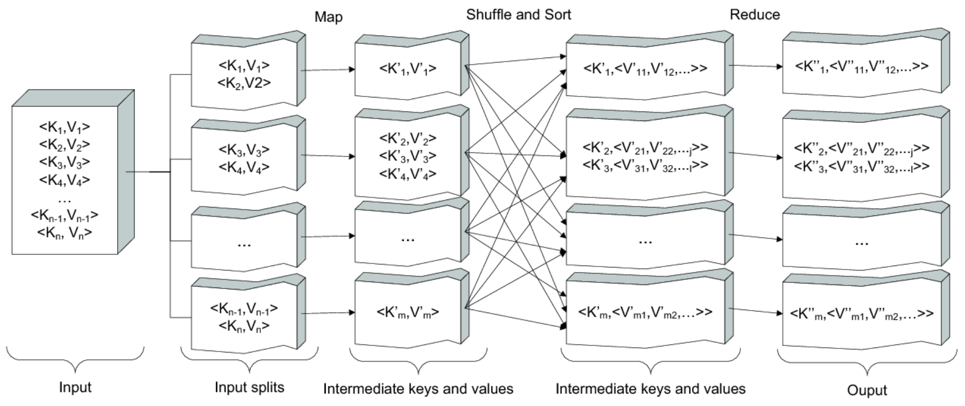


Figure 2.6: The MapReduce framework.

2.2.3 MapReduce programming framework

The Cloud Computing solution, however, just enables the flexible and scalable infrastructure to deal with storage and computation issues. To deal with performance and scalability when processing a huge amount of data, we need to have a special parallel programming model. Recently, Google has designed a parallel computing framework called Mapreduce [49] which can scale efficiently many thousands of commodity machines. These commodity machines forming a cluster can be accessed by users in an institution or can be rented over the Internet through utility computing services. Actually, the idea of this framework is not new since it has already been used in traditional functional programming languages such as Haskell, Lisp, Erlang, etc.

The basic idea of the MapReduce framework is shown in Figure 2.6. The data that need to be processed is divided into "input splits". Each split contains many records in a key, value pair structure. The map blocks (defined by software developers based on the application business) map these input key, value pairs into other intermediate key, value pairs. This intermediate data is then sorted and grouped together based on the keys. As a result, the input of the reduce blocks is a key with a collection of values. The reduce blocks (also developed by MapReduce programmers) then produce the final results in the form of key-value pairs as well. One very important feature enabling MapReduce to process a huge amount of data efficiently is that all maps and reduce blocks are executed concurrently. There are two main phases though: map and reduce. As we can see from the figure, all map tasks need to finish before running any reduce tasks.

There are many different implementations of the MapReduce framework such as Hadoop, Phoenix, Disco, Mars, etc. In developing our tool, CloudAligner, we chose Hadoop (http://hadoop.apache.org) since it is open-source (easy to fine tuning), written in Java (high portability) and widely used in both academy and industry.

CHAPTER 3

GENOMICS APPLICATIONS

After knowing related background of Genomic research and Cloud Computing, in this chapter we are going to verify our hypothesis that MapReduce is a very promising programming model to process the genomic data as detailed in Section 1.4 of Chapter 1. Figure 3.1 summarized the basic idea of this chapter. On the left hand side of the figure represents the traditional approach in the view of layers. The right hand side of the figure is layers of the modern approach. The new tool we will talking about shortly, CloudAligner, is also shown its position in that figure.

In this chapter, we will focus on the fundamental application of genomic: sequence alignment. Almost all sequence analysis in genomic have to take place after finishing this step.

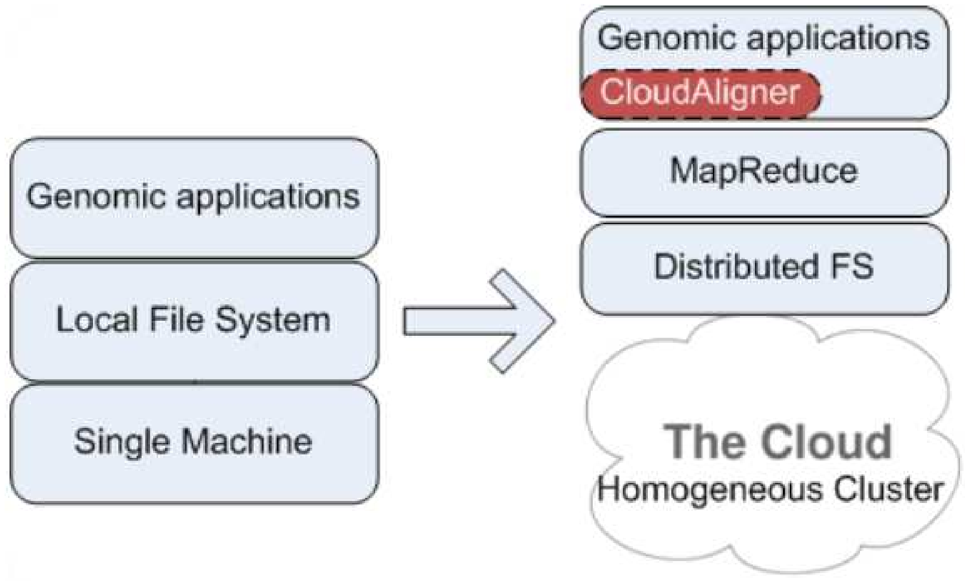


Figure 3.1: Application.

3.1 Current scenario

The rapid development of new sequencing technologies helps improve the accuracy as well as scope of many biological applications such as the assembly of the genome or transcriptomes, ChIP-Seq, RNA-Seq, etc. Most of these applications execute the read alignment as their first step. Therefore, the sequence alignment is the most important and fundamental part to almost all applications of sequencing analysis. The extensive genetic informational datasets create many serious problems and challenges for the popular alignment tools such as bowtie [81], RMAP [126] [125], MAQ [84], bwa [87][85], etc.

There have been some initiatives toward the trend of using MapReduce in sequence alignment such as CloudBurst [117], SeqMapreduce [88], Crossbow [82], etc, and the results are very promising. These tools can provide better performance and friendlier (web-based) user interface than local approaches.

However, in spite of these promising features, this is just a new approach. Therefore, the functionality of these existing cloud-based tools is limited. They do not offer variety features as well as the friendly interface needed to popularize them. For instance, the common functions that are often implemented in well-established on-premises alignment tools are bisulfite and pair-end mapping since detecting genome variations such as single nucleotide polymorphism (SNP) or large-scale structural variations is a very important in biological analysis. The CloudBurst, for example, doesn't support either of these features. It also doesn't support the fastq input format which is a very common output of current sequencers. In addition, its interface is a command line style which is not very friendly. Another MapReduce based software, SeqMapreduce, is a performance improvement version of CloudBurst, but its website and code are even inaccessible now. Crossbow is the read mapping and SNP calling software that run in the Amazon EC2 cloud. It consists of a set of Perl and shell scripts that allow Bowtie and SOAPsnp to run on Cloud. It actually has a very nice and friendly web interface created with the aid of JotForm. However, since its bio functionalities depend entirely on other tools (Bowtie and SOAPsnp), it inherits their shortcomings too. For example, Bowtie can only allow at most 3 mismatches in its mapping and was only designed for short reads. Therefore it can't improve or fine tune the core functional algorithms.

Consequently, we developed CloudAligner to address such limitations of the existing tools and also to advocate a Cloud and MapReduce-based solution for genomic problems. Especially, CloudAligner is designed to achieve better performance, longer reads, and extremely high scalability. It has more common functions such as bisulfite and pair-end mapping as well as a friendly user interface, and it supports more input as well as output formats.

3.2 Software Design

Figure 3.2 shows the overall architecture of our tool. Not following the traditional MapReduce model 2.6 like most other tools, CloudAligner does not have the reduce phase. The mapping algorithm (the popular seed-and-extend alignment algorithm) is implemented entirely in map tasks. By doing this, we don't have to spend time on operations such as shuffling and sorting of intermediated data. Also, parts of the final results can be obtained with this method as long as at least one map task successfully finishes since each map task aligns a small set of read on the whole genome. It is completely independent from other map results. This property is very beneficial especially for time-consuming jobs and the "pay-as-you-go" model of the Cloud because when the job fails, we can still have part of the result and only need to re-execute and pay for mapping the failed parts again.

There are two main input files for CloudAligner as the other alignment tools: the reference file and the read file. The reference files are normally in the fasta format while the read files can be in the fasta or fastq format. Both are changed into the serialized files (to be easily processed over the network) and copied to the HDFS or Amazon S3.

When executing, CloudAligner cuts the read file into smaller chunks called input splits (each read contains many read sequences) and distributes them to the mappers. Each mapper aligns its input split onto the the whole reference genome file.

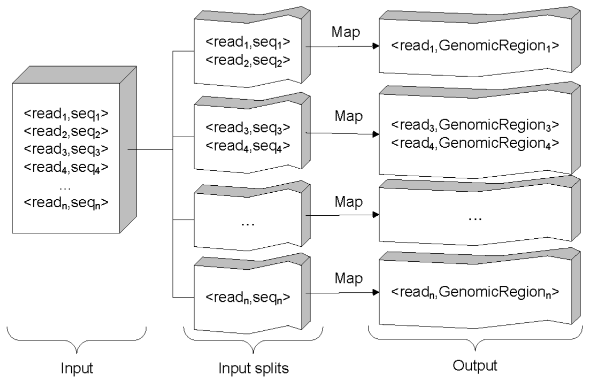


Figure 3.2: CloudAligner architecture.

3.3 Experimental Results

Evaluation criteria

We are going to evaluate CloudAligner in term of performance and accuracy. The performance metric is actually measured as the execution time of the tools. The accuracy is the number of reads that are mapped uniquely on the reference genome. To measure these two metrics, we built a Hadoop cluster of 13 nodes as a testbed for our experiment. The configuration of machines in our testbed is shown in Table 1. In the following experiments, the time to convert data to the Hadoop format and the time to move them into Hadoop Distributed File System (HDFS) or Amazon S3 are excluded.

Mapping performance

As CloudBurst is also a Hadoop MapReduce based alignment tool, and the CloudBurst's paper [117] has shown the performance improvement (in term of speedup) over RMAP, we would like to compare our performance with it only. It is noteworthy that in our map task, we adopted the seed-and-extend mapping algorithm with different patterns in the seeds like RMAP. However, CloudAligner was developed in Java and doesn't have the limitation of using 64bit machine as RMAP.

In this experiment, we ran both CloudAligner and CloudBurst on the same system with the same data set. The data set is obtained from the CloudBurst website. Figure 3 shows the per-

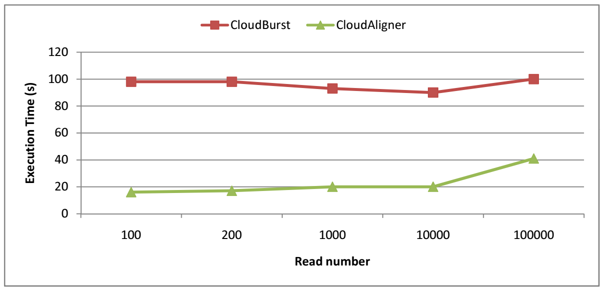


Figure 3.3: Performance of CloudBurst and CloudAligner.

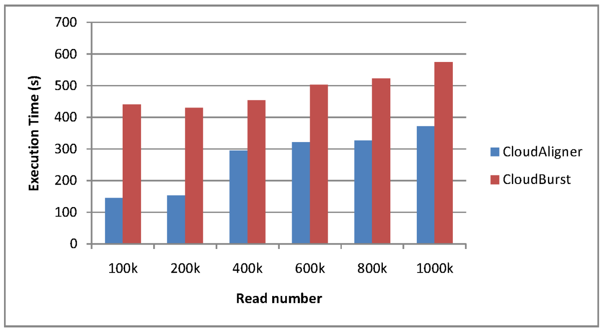


Figure 3.4: Performance of CloudBurst and CloudAligner on larger data.

formance results of both types of software with a different number of reads (the same reference file). The x axis in the figure is the number of reads, and the y axis is the execution time in second. From the figure, we can see that CloudAligner is 60 to 80% faster than CloudBurst.

We also did another experiment on the real data from the 1000 Genomes project. In particular, we mapped different subsets of the accession SRR035459 to the human chromosome 22 (50Mbp) allowing up to 3 mismatches. The results of this mapping is shown in Figure 4. From the figure, we can see that the execution time of both CloudBurst and CloudAligner is proportional to the number of reads, and CloudAligner outperforms CloudBurst from 35 to 67%.

3.3.1 Mapping accuracy

Since CloudAligner also employed the popular seed-and-extend algorithm like RMAP and Cloudburst, it generally inherits the limitations of this type of algorithm. Basically, this approach trades the accuracy for the performance. Instead of comparing the whole reads (with mismatches), the algorithms of this type only search for the shorted sequences called seeds. The accuracy of the result depends heavily on the seeds. The seed alignment can be consecutive or non-consecutive (template, pattern) matches.

To verify our results, we ran both CloudAligner and RMAP on the same set of data with equivalent seed information. With this type of experiment, we don't need to choose a very large workload because we only focus on the accuracy of the results. First, we ran CloudAligner with all the appropriate test reads (single-end, bisulfite, pair-end, fastq reads) of RMAP. Each data set has 100 reads with 25 bases in length. Our output files are the same as those of RMAP though with a different order. Second, we would like to test CloudAligner with another larger data set and longer reads to strengthen the soundness of our results. This time, RMAP and CloudAligner were executed (in mismatching mode) on the data includes 100,000 of single-end reads, and the reference genome is of the Streptococcus suis. Each read has 36 bases. CloudAligner only identified 74,208 unique maps while RMAP produced 74,291 unique maps. After carefully examining the extra 83 reads, we found that RMAP doesn't count the bad bases in the reads as mismatches which we should. Therefore it found more results with the same number of allowed mismatches.

3.3.2 CloudAligner in Amazon EC2

To experience how our tool behaves in the real Cloud, we uploaded it to Amazon simple storage (S3) and created job flows in Amazon Elastic MapRreduce to execute it. In addition to normal arguments such as read length, reference genome, input, output locations, CloudAligner (like other MapReduce applications) has the number of maps and reduces parameters. In this part, we would like to study the effect of choosing different number of maps on the performance because, in our approach, there's no reduce task. Figure 5 shows the execution time of CloudAligner in the Amazon EC2 when mapping 2 millions of reads on the human chromosome 22 with different number of maps. The experiment was performed on 20 small EC2 instances. Thus we have totally 38 map slots (1 instance is used for the master node). The

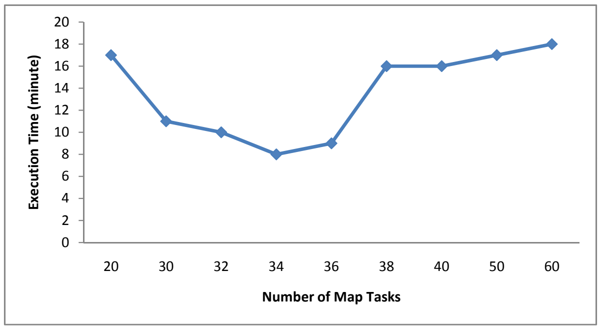
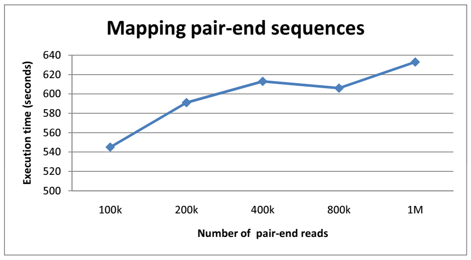
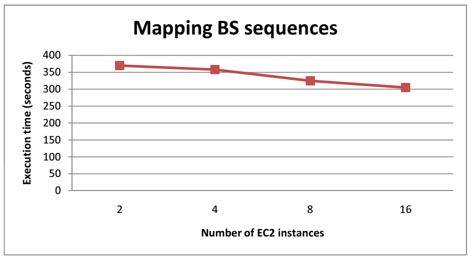


Figure 3.5: Performance of CloudAligner in Amazon EC2.



(a) Pair-end mapping



(b) BS mapping

Figure 3.6: Pair-end and Bisulfite mapping in EC2.

information in the figure suggests that the optimal number of input splits (maps) should be a little bit less than the maximum number of map slots. In this case, it should be either 34 or 36.

The performance of pair-end and bisulfite mapping functions of CloudAligner is expressed in Figure 6 and 7 respectively. The pair-end read data is 76bp in length and was obtained from the results of sequencing the African honey bee sample in our lab. Figure 6 shows the execution time when mapping different numbers of pair-end sequences (with quality scores) onto honey bee's chromosome 1 (A_mel 4.0). All of these mapping were taking place on 20 medium EC2 instances of Amazon Cloud. With the same number of instances, processing more read requires more time. For the BS mapping demonstration, we used the 100k synthetic reads from BS Seeker. Figure 7 shows the execution time of this type of mapping with different number of EC2 small instances. Intuitively, the more instances we throw in, the faster the program is.

Table 3.1: the configuration of our main Hadoop testbed.

Type	**Machines #**	**CPU**	**Memory**	**HDD**	**OS**
Server	1	4 cores AMD 2GHz	6GB	250GB	64 bits Ubuntu Server 9.04
Server	12	1 core Intel Xeon CPU 2.80GHz	4GB	40GB	64 bits CentOS

Table 3.2: The local Hadoop testbed configuration for the heterogeneity experiment.

Machines #	**CPU**	**Memory**	**HDD**	**OS**
7	1 core Intel XEON CPU 1.80GHz	512MB	160GB	32 bits CentOS
21	1 core Intel Pentium III	512MB	20GB	32 bits Ubuntu 8.04

Table 3.3: The features of CloudAligner and other closely related tools.

	CloudAligner	CloudBurst	RMAP
Mismatch Mapping	✓	✓	✓
Bisulfite Mapping	✓		✓
Pair-end Mapping	✓		✓
Fastq input	✓		✓
SAM output	✓		
Executable in Cloud	✓	✓	

3.3.3 Discussion

In general, in terms of performance, CloudAligner outperforms CloudBurst and RMAP. The performance gain over RMAP is mainly based on the scalability and parallel processing. With CloudBurst, the limitation of its approach is the network bandwidth. With CloudAligner, its limitation is in the computation power of the workers in Hadoop. Consequently, if we run CloudAligner on cluster of legacy machines with high speed network, we probably loose the performance advantage over CloudBurst. However, as shown in Table 1, the machines in our cluster are also not powerful at all. All of them only have a single core. Moreover, the interconnection between them is a brand new high-speed network (1Gbps) since we put them in our newly built server room. Therefore, it is safe to conclude that in common cluster CloudAligner generally performs better.

We also developed the web-based interface for CloudAligner and hosted it at http://mine .cs.wayne.edu:8080/CloudAligner/. From the website, users can upload the reads as well as the reference files in text format. The upload servlet automatically translates them into the Cloud format and uploads them to our Hadoop cluster. After having the files in the system, users can select them for the mapping together with common parameters such as the number of mismatch, seed, output format and so on. After finish mapping, the website creates a link to download the results.

CloudAligner can easily run on heterogeneous clusters. There are no restrictions on the hardware configuration of the machines constituting the cluster as long as they have enough memory to handle the small chunk of reads and reference genome assigned to them. To demonstrate this ability, we built a cluster of outdated machines as shown in Table 2 (except the master node) and ran CloudAligner on it to map 2 million reads on the human chromosome 2 (237Mbp). It took 27 minutes and 18 seconds to finish this job. The only minor adjustment we need to do to handle larger data sets on outdated machines is to periodically inform the Hadoop system that our tasks are still alive. Otherwise, it assumes the nodes are dead and initiates the tasks on other nodes. This default time out is 10 minutes in Hadoop.

CloudAligner also offers the option to produce output files in both SAM [86] and BED6 formats to enable easier post processing analysis. For example, biologists can use the samtools [86] to identify SNP or INDEL in their samples or convert to the BAM file to have a visual view of the alignments.

Although CloudAligner theoretically has no limitations in read length as well as in number of mismatches, to efficiently deal with long reads, we should apply additional methods on the seeds such as the two-level techniques of Homer [67].

3.4 Summary

With the improvement in sequencing technology, the data generated by the sequencers is becoming cheaper and better. Therefore, more data is increasingly being generated which leads to serious issues in storing and processing. Combining Cloud infrastructure and Mapreduce framework together is emerging as one of the best solutions. However, the current tools of this trend are lacking the common features found in other popular tools making them unattractive to the users.

In this chapter, we've built CloudAligner with the most common functions required for a mapping tool as well as an easy-to-use web-based interface to endorse the tendency of using Cloud and MapReduce. The summary of these functions is described in detail in Table 3.3. Moreover, we also designed and implemented a new approach to improve the performance of our tool. Our results indicate that significant improvement in the performance of alignment MapReduce-based tools can be achieved by omitting the reduce phase.

CHAPTER 4

OPERA: OPEN REPUTATION MODEL

Now we knew that MapReduce framework and Cloud infrastructure are very efficient to process the huge data in genomic. It's time to take a deeper look at the infrastructure. The questions we target here include what type of infrastructure that is suitable for Genomic applications? What requirements for or characteristics of a success infrastructure? What are challenges to build such system? In this chapter and the next one, we mainly focus on the heterogeneity.

Generally, in term of computation, the system is expected to have high performance, throughput, and efficiency. It needs to be scalable too. In terms of storage, availability, reliability and security are the main concerns.

The underlying infrastructure can be a single powerful server, cluster, grid, P2P, Cloud (with many VMs). At this time, most biological organizations are still using a single powerful server with many cores, RAM and HDD to store and analyze their data. While this approach is simple, secure and easy to manage, it has many drawbacks. It's not scalable and efficient. The PIs need to invest a large amount of money to the up-front cost of such machines. They also need to estimate the computing ability and storage capacity beforehand in the budget of their proposal. This is an error-prone process which may lead to resource inefficiency due to over or under-estimated resource provision. Cloud platform with its elasticity appears to be a good infrastructure here. With the Cloud economic model, the users only need to pay for the resource they actually consume.

Taking a look deeper inside of the Cloud, we found that it is actually a heterogeneous system. This heterogeneity is not suitable for the Hadoop MapReduce framework of which the basic assumption about the underlying system is homogeneous. This violation causes a significant degradation in the performance of Hadoop. Therefore, we offer a new reputation-based scheduler to tame this heterogeneity in this chapter.

Like the previous chapter, Figure 4.1 provides the summary of this chapter. The layer view of the system we start with is on the left of the figure. It is the system we have considered so far. The layer view on the right of Figure 4.1 expresses the detailed look of the system. It shows the works and layers we are going to explore in this chapter: the reputation-based scheduler and OPERA.

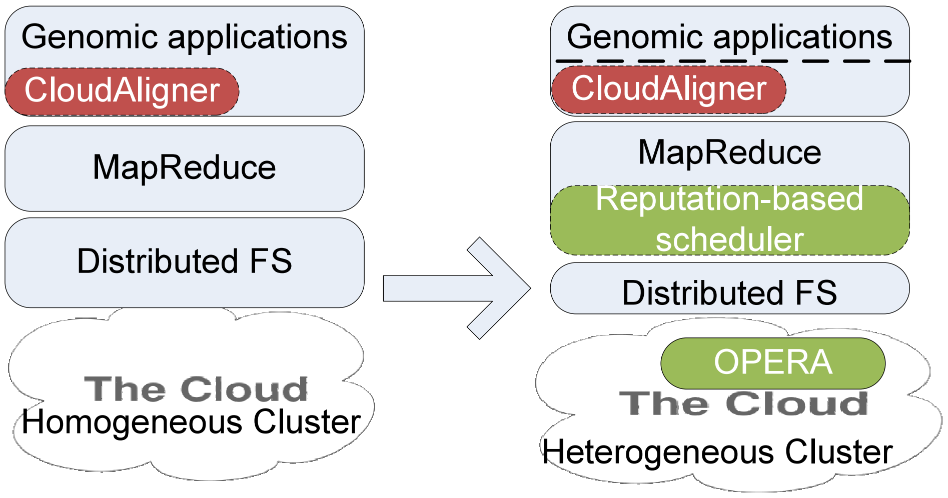


Figure 4.1: System with Heterogeneity Consideration (OPERA).

It is noteworthy that this chapter and the following chapter are closely related. When describing DiR in Chapter 5, we mention Opera. In other words, DiR is the user of Opera.

4.1 Introduction

With the increasing demand in computing, the number of data centers is also increasing [44]. However, according to the EPA report in 2007 [54], the data centers are going to consume more than 100 billion kWh by 2011 and account for 23% of global information and communications technology (ICT) CO_2 emissions to the environment [10]. As a result, making the operation of Data Centers “greener” has become the main focus of many research activities recently. The main idea is to consume energy in an efficient way since current systems are efficiently [28, 33, 26]. First, let us take a look the resource inefficiency in current systems.

4.1.1 Resource Inefficiency

The inefficiency is caused by the waste. There are several types of waste at different levels such as infrastructure-, machine- and system-level. At the infrastructure level, one half of energy is spending on cooling [74]. At the machine level, 50% energy is used during the idle [28]. At the system level, the system could be useless because of checkpointing [119] or context switching [66]. To improve resource efficiency, we have to minimize these waste.

To measure the data center energy efficiency, some metrics have been proposed. Essentially, energy efficiency of a data center is defined as the amount of computational work performed divided by the total energy used in the process.

$$\text{Energy Efficiency} = \frac{Computation}{\text{Total Energy}}$$

Recent work [41, 47] has defined two widely used metrics (Power Usage Effectiveness (PUE) and Data Center Infrastructure Efficiency (DCiE)).

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} \tag{4.1}$$

However, these metrics only capture the efficiency in data center at infrastructure level. Therefore, Barroso and Hölzle [26] have proposed another way to compute efficiency which takes into account different levels of efficiency as shown in Equation (8.6). In the formula, energy efficiency is factorized into three parts. The first two factors account for the efficiency of power dissipation to electronic components in a system. SPUE is server PUE which is calculated in the same way as PUE (see equation 4.1), but its denominator only accounts for power consumed by the electronic components directly involved in the computation such as motherboard, CPU, memory and so on. The last factor represents the efficiency when running workload in the data centers.

$$\begin{aligned}\text{Energy Efficiency} &= \frac{Computation}{\text{Total Energy}} \\ &= \left(\frac{1}{PUE}\right) \times \left(\frac{1}{SPUE}\right) \times \\ &\left(\frac{Computation}{\text{Total Energy to Electronic Comp.}}\right)\end{aligned} \tag{4.2}$$

$$\begin{aligned}\text{Energy Efficiency'} &= \left(\frac{1}{PUE}\right) \times \left(\frac{1}{SPUE}\right) \times \\ &\left(\frac{Computation}{\text{Total Energy to Electronic Comp.}}\right) \times \\ &\left(\frac{\textbf{Required Resource}}{\textbf{Actual Resource Used}}\right)\end{aligned} \tag{4.3}$$

While this definition is good enough to characterize the first two levels of waste, i.e., infrastructure- and machine-level waste, we argue that it is not the end of the story since the execution of a job (i.e., run-time efficiency) is not considered. For example, a job will have to be rescheduled or resubmitted if it fails in the middle of the execution, which will cause system-level waste. We augment the Equation (8.6) by adding another definition of **usage efficiency**, which is defined as $\frac{\text{Required Resource}}{\text{Actual Resource Used}}$ as shown in the Equation (4.3).

Our work focuses on reducing the waste caused by failures because future systems will be expected to have higher failure rate [119]. We will show that the failure does cause increasing in energy consumption and that reducing the failure indeed helps to reduce it. Hence, our objective is to minimize the number of job resubmission, i.e re-execution, while still maintaining the performance. Basically, this type of waste comes from inefficient resource allocation. For example, if the resource allocator assigns slow, unreliable nodes to handle critical tasks, the whole application can easily be delayed or even fail. As a result, we need to choose proper resources to execute our tasks to achieve the goal.

4.1.2 Our Approach

Selecting appropriate resources is not new. In commercial systems, the consumers choose the service providers (SPs) by their reputation. In computational systems, resource selection selects the slow-progress node for speculative execution based on their progress score [139], a group of nodes for computational replication based on their reputation [127], or the locations to store replicas [79].

However, in general, the existing methods rely on a single-value reputation to capture the differences (the heterogeneity) between the service providers in terms of a property such as performance or availability. However, this single value representation is inadequate for the **multi-property based selection criteria**. For example, there are conflictions, such as the one between data locality and fairness as shown in [138], that make it unsuitable to apply conventional single value based scheduling. In addition, the reputation computed from the existing models is generally based on the past information, which may be inconsistent with the current status of the system. For example, if a node has had a good record for a long time and now it becomes unreliable because of a virus or recent compromise, the history-based computation model would still return a high reputation.

As a result, we use a vector instead of a scalar to represent the reputation and capture different concerned properties. For example, with a reliability concern, an SP has a higher reputation if it has more *successful* transactions than others. With the performance concern, the higher the reputation, the higher performance a resource has. We developed a new trust model called *Opera* (OPen ReputAtion Model) that allows users to query the reputation vector of any registered component. Opera also allows the addition of new elements to the reputation vector, which expresses its *openness*. To reduce management intervention, Opera employs a popular, open-source monitoring tool to capture the changes of the system and dynamically update the reputation vectors accordingly. Based on the reputation vectors, the system can easily select resources that conform to the users' requests.

To illustrate our idea, we modified an existing scheduler, which can be considered as a resource allocator, to leverage Opera. The chosen scheduler is the default scheduler of Hadoop, a popular framework for running MapReduce [49] applications on large scale clusters of commodity machines [8]. We conducted the experiments by executing different MapReduce applications with different configuration requirements on our local cluster, under the presence of failure and heavy-load nodes. The results show that the original Hadoop, although built with failure toleration, can still suffer from failures and increases the execution time to 50%. Moreover, the energy consumption of the whole system can increase 17%, too. The results also confirm that our modified scheduler actually selects appropriate nodes and, therefore, improves the performance up to 32%, reduced up to 59% of the number of failed/killed tasks as well as improve the (energy) usage efficiency by up to *53.32%*.

The contributions of this section are three-fold: (1) a new reputation model including *vector representation* and *just-in-time feature* (JR); (2) the design and implementation of Opera; and (3) the incorporation of it with Hadoop and comprehensive experiments.

4.2 Background and Related Work

Reputation systems give information about the past behavior of an entity, helping one to decide which entities to trust. Often reputation is used in commercial systems in which the entities are SPs and customers, and in P2P systems where peers are the entities. As a customer wants to do her business in new environments where she has no prior experience with the SPs, a conventional way is to rely on the reputation of the SPs or their history behaviors. Next, we will give a brief overview of how reputation has been calculated and used in scheduling in previous work.

4.2.1 Reputation computation

In this section, based on the previous work of Sonnek [128] and Liang [94], we briefly review some existing reputation models before introducing their applications. For convenience, let R be the reputation of a node. It is worth noting, to our knowledge, R that has been always defined as a scalar in previous work. Different approaches focus on how to calculate this value from multiple inputs. In this section, as we will see in Section 4.5, we propose to use a vector to represent reputation.

Ebay's rating mechanism: Ebay, a popular online auction and shopping company [9], uses a simple yet efficient feedback mechanism to compute reputation. Their model is considered the most popular and successful to date. In their model, after using a service or finishing a transaction, users are allowed to rate the SP. In particular, for the i^{th} transaction, a user's rate can be calculated as follows

$$r_i = \begin{cases} +1 & \text{if she is satisfied} \\ -1 & \text{otherwise} \end{cases}$$

and the reputation of the corresponding SP is

$$R = \sum_{i=1}^{\text{total executed tasks}} r_i$$

Beta rating: This model was proposed by Jøsang and Ismail [46]. It is based on the beta density function governed by two parameters, α and β. The feedback of client X giving to the SP T is expressed by the function

$$\varphi\left(p \mid r_T^X, s_T^X\right)$$

in which r_T^X and s_T^X are the degree of satisfaction and dissatisfaction of X about T, respectively. The reputation of T is also the probability expectation value

$$R = E\left(\varphi\right) = \frac{(r+1)}{(r+s+2)} \tag{4.4}$$

In the bootstrap state, both r and s are equal to 0; therefore, we have $R = 0.5$. Note that $r + s$ is the total amount of feedback so far. From equation (4.4), we can see that as the amount of satisfied feedback increases, the reputation increases, which is intuitively reasonable.

Weighted rating: This method was proposed by Azzedin and Maheswaran [25]. In this method, they differentiate two important concepts: direct experience and reputation. Each of them has a weight in the reputation formula. The reputation value R is computed as follows:

$$R = \alpha S + \beta F \, (\alpha, \beta \text{ are the weight values})$$

where

$S = \frac{r}{(r+s)}$: the self experience of a node when it communicates with the SP. (r and s are the number of satisfied and dissatisfied transactions, respectively)

$F = \frac{(\sum r)}{(\sum r + \sum s)}$: the experiences of other clients when using that SP's service.

In addition, there are other models that take into account risks [93], employ recommendations [75], and filter bad raters or abnormal ratings [19], and so on.

4.2.2 Reputation-based scheduling

There are many applications for reputation, but we only focus on applications regarding scheduling. Our study was inspired by the recent work of Matei *et al.* [139] and Sonnek *et al.* [127] in the sense of using reputation-based scheduling. Matei *et al.* showed that the MapReduce implementation of Hadoop [8] suffered a significant decrease in performance in heterogeneous environments; therefore, they proposed a LATE scheduler that is based on a new progress score computation method. These progress scores, which are only valid within a specific job execution, can also be considered as the reputation of a node. We also chose Hadoop as our evaluation framework because we want to tame the heterogeneity that it is suffering from. Apart from Matei's work, we modify its scheduler to leverage Opera instead of proposing a new computation of the progress scores.

Sonek *et al.*[127] also adopted reputation-based scheduling, but they exploited reputation to decide the size of a group in that if we schedule the same task to each member of that group, we are likely to obtain the correct result thanks to the majority vote. Intuitively, the group size should be small if its members have high reputation and verse versa.

Another closely related work is that of Alunkal *et al.* [22]. They also proposed a reputation-based resource selection service for Grid, which was based on [77] and [25], but they didn't have any real implementation for the proposed service. Generally, it is difficult to evaluate a work based on trust or reputation because of the lack of real feedback data from users.

Besides, as a resource selector, our work is also similar to Nimrod/G[36], Condor-G[60] and Matchmaker[109]. However, Nimrod/G employed computational economy which used one-value (cost) to represent the resource. Users specify price and deadline, Nimrod finds resource based on the matching between cost and price. It doesn't care about the quality of the resources and the reputation of resource providers/sellers. All three of them focus on how to match the resource requirement to resource status before the execution of jobs. They didn't mention about during the execution time of the jobs. In addition, Opera considers from both the systems and application perspecitve (e.g data locality), while Matchmaker focuses on the system perspective.

4.3 Opera Design

In this section, we argue that it is insufficient to assess an entity via only a scalar value as in previous methods. Existing reputation systems such as eBay[9] use this successfully because they are only concerned about one behavior (successful buy/sell transactions) of an entity. As long as users have many successful transactions, they will have a high reputation in the "eBay world". It doesn't matter how good or bad they are in the other *worlds* (or contexts). The more information we have about the entities, the better decision we can make when selecting them. In addition, as we focus on scheduling systems, in which the entities are nodes and scheduler, by capturing many different aspects of a node, the system can offer different types of services to users, such as highly available service, powerful computation service, secure service, and trust service, to name a few. Moreover, mapping many components into one value as used in beta or weighted rating does not provide enough flexibility in the selection criteria. Therefore, in our new *Opera* model, each entity or node has a *vector* of reputation instead of a single scalar value. Actually, this representation is somewhat similar to the ClassAd [109] in the Grid Computing world.

In this section, we give a general design of such a model by answering three questions: (1) Who is going to use our model? (2) What services will our model provide or what are the functionalities of our model? and (3) How can we leverage it?

4.3.1 Opera Users

Three types of users are involved in the operation of Opera, including *ratees*, *usual clients* and *reputation system designers*. *Ratees* are those who need to be rated or for whom the reputation is computed. They can be a machine, a service or a service provider. The *Usual clients* are the customers of the *ratees*. They may be called *raters* if they provide rating information or *feedback* to the system. The *usual clients* can also be the end-user or other components in the system such as the scheduler or resource manager. For example, the scheduler may use Opera to choose suitable nodes in order to improve the performance or the throughput. The resource manager may use Opera to improve the utilization of the system. *Reputation system designers* are researchers who define how to compute the reputation.

4.3.2 Opera Objectives and Approach

Ultimately, Opera provides a service that helps clients select a set of suitable candidates, among available resources or services that meet their requirements. In order to do that, Opera first gathers as much information about the ratees as possible, then calculates the *reputation vectors*, and finally selects candidates that have reputation elements agreeable to the user's *criteria*.

The information about the ratees is obtained either from inside the system or the feedback provided by the raters. In other words, the judgment is enabled from both the system and the user's point of view. The information inside the system is collected by *agents* installed at each component. With coarse granularity, these agents can be the user-developed software or built-in services such as the “SNMP” (Simple Network Management Protocol) on a machine. With finer granularity, these agents can be the sensors attached on specific components of the machine. Consequently, Opera needs to employ an extensible monitoring tool as an *internal rater* and have appropriate APIs to receive rating information from *external raters*. By doing this, Opera is equipped with self-adaptability, meaning that it doesn’t need any management intervention, which is crucial in unsupervised open environments. For example, a resource manager is capable of feeding the monitored information to statistical machine learning algorithms to achieve self-optimizing [101] in resource utilization. Further, since the extensible monitoring tool can dynamically add new monitoring metrics, it also contributes to the *openness* of Opera by facilitating the Opera users, e.g., researchers in the reputation community, to collect any information they want.

Users can either provide feedback (rating information) to Opera to use reputation built from these feedback or simply use the rating information provided by agents. If users are ratees, i.e. they provide feedback, there are many known issues, such as selfishness, maliciousness, collusion, badmouthing or ballot-stuffing clients, that need to be solved but are out of the scope of this work. However, as users also need to register with Opera to receive service, they are able to use this information to support models that deal with these listed issues. For example, one can weigh the clients to decide their effect on the reputation of SPs [95].

Using the collected information, Opera calculates the reputation values based on both *pre-defined* and *user-defined* models of computation. The predefined models are built in as a part of Opera and are mainly used by usual clients. The user-defined models are developed by reputation system designers, and this feature is another aspect of the *openness* of Opera. To enable this extendibility as well as to capture the heterogeneity of ratees, Opera uses a **vector structure** to store the reputation elements calculated by different models. Each reputation value is an element of the reputation vector and represents a specific point of view about the ratees.

For instance, as seen in Section 4.5, we define the reputation R as $< J_{app}, J_{sys}, H_{app}, H_{sys} >$, in which J and H represent the current (Just-in-Time) and past behaviors of the system; app and sys denote the application and system point of view, respectively. When reputation system designers want to develop a new model, they add a new element to this reputation vector and provide a method to compute it. The usual clients most likely choose a predefined model that best fits their needs. One may ask these questions: Why do we need to calculate reputation? Why don't we use the collected information directly to select resource? This is because some models may require more than single monitoring information in their computation.

After getting the reputation of each ratee, the users (either usual clients or reputation system designers) specify the criteria to select the resources or use the predefined ones. It is this ability that enables the **diversity of a service**. The criteria may be based on only one reputation model (an element of the reputation vector) or a combination of many elements. In addition, they also have different types of constraint. For example, a criterion may be choosing *"top three nodes that have the highest availability," "nodes that have availability greater than or equal to 3 nines," "nodes that have availability higher than the average availability," "nodes that have above average availability and below average load"* or *"two nodes that have the smallest number of failed tasks,"* etc. Note that in these examples of criteria, availability, load, or the number of failed tasks is considered as a model of reputation.

To summarize, Opera collects the information both from inside and outside of the system, calculates the reputation of the system from it based on predefined and user-defined computation models, and selects candidates based on predefined or user-defined criteria. The main APIs of Opera are summarized in Table 4.1. It is the user's responsibility to choose which model and criterion are suitable for them as usual clients, or they can define their new model to compute reputation as reputation system designers. This also means that we don't provide solutions to the traditional problems such as how to know if a rating is correct, how to force a client to provide rating information, how to avoid the fact that an SP performs very well at first to obtain high reputation and then turns bad or malicious, how to maintain fairness between newly joined SPs and the old ones, and so on. We envision that the answers to most of these questions are domain-specific and need to have domain knowledge to address them. Resilient reputation management is still an open problem [98].

Table 4.1: The Opera's APIs.

Application Programming Interfaces	**Description**
`set_time_window_size(duration)`	set the size of the time window in which we calculate the reputation
`register(host,port)`	register newly joined service providers (SPs)
`subscribe(host,port)`	subscribe by usual clients or raters
`rate(rater_id,ratee_id,trans_id, time, value)`	rate a transaction of a SP
`getReputationLength()`	get the current length (number of dimensions) of the reputation vector
`getRateesList()`	get list of ratees or SPs
`addNewDimension(startupValue)`	add a new element to the reputation vector
`getReputation(rateeID,dim)`	get the specific reputation of a specific node
`setReputation(rateeID,dim)`	define new model of computation

4.4 System Architecture

The overall architecture of the system is shown in Figure 4.2. From the figure, we can see how Opera communicates with the existing system. Opera applies the client/server model in which the server (the circle named Opera) communicates with 2 clients represented by 2 circles named Service Providers and Clients/Rater. However, Opera only manages the reputation of the registered SPs only. This model was chosen because it is simple and provides a global and consistent view of the system. Applying this centralized approach, one may challenge Opera's scalability and availability. Actually, they depend on the implementation; therefore, it will be discussed later in Section 4.5.

4.4.1 Assumptions

Before going to the implementation, it is useful to clarify all the assumptions we made.

The first assumption is that the future behaviors of a node can be inferred from the past be-

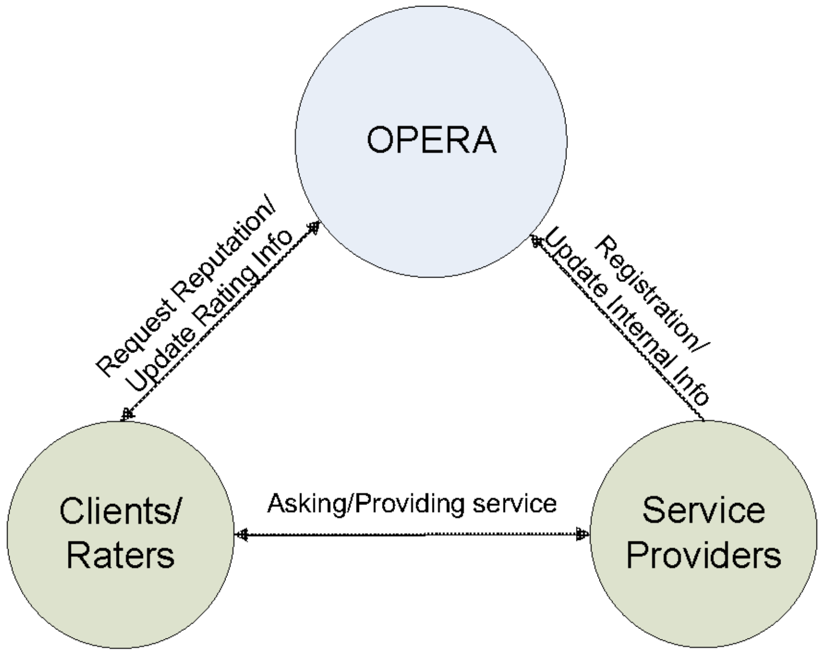


Figure 4.2: Opera system design.

haviors, which is the basic assumption for many other research fields such as machine learning and data mining.

The second assumption is that the Opera server, which is responsible to calculate and store the reputation values of all participating nodes, is secure and trustworthy to simplify the design. This assumption is reasonable given the fact that data-intensive computing systems like Hadoop (which will be used as a specific application scenario in our implementation) often reside in Data Centers which are highly secure.

4.5 Materialization and Implementation

To evaluate Opera, we implemented Opera using Java and modified the existing scheduler in Hadoop (version 0.20.0) to leverage it. However, we focus only on selecting nodes, *not the time*, to assign tasks because the scheduler of Hadoop is using the *pulling model*, which means as soon as a worker is free, it contacts the master to ask for tasks periodically. The time to assign the tasks cannot be decided by the scheduler. In this case, the usual client of Opera is the Hadoop scheduler, and the ratees (SPs) are nodes in the cluster.

Normally, reputation is often used in the world where participants can cooperate without knowing each other in advance. Nevertheless, in this work, we chose Hadoop, which is an open-source platform being widely used in data intensive computing, to interact with Opera because we would like to demonstrate Opera applicability in different scenarios. As another reason, we want to show that although Hadoop is a highly fault-tolerant system, we can still improve its robustness with Opera.

As in the design part, Opera also needs to employ a monitoring tool to observe the internal system behaviors (beside the feedback of raters), and we chose *Ganglia* [7], an open-source scalable distributed monitoring system, to fulfil this task.

Generally, in manipulating the reputation data, Opera applies the Share Memory model with the Producer-Consumer mechanism. The producers are the threads that implement the predefined or user-defined reputation computation models. Each thread corresponds to one model. It computes the reputation and periodically updates the corresponding element in the reputation vector. The consumers are usual clients who use reputation elements in their criteria to select candidate SPs.

In this section, we are going to show the detailed computation of each element in the reputation vector of Opera, the selection criteria, what modifications we did on Hadoop to connect with Opera, and the scalability as well as the availability of the model.

4.5.1 Reputation Calculation

After setting up the system, we are now able to define the reputation vector quantitatively. As mentioned in the design section, we recommended a **four-element vector** for the reputation. Let vector $R =< J_{app}, J_{sys}, H_{app}, Hsys >$ be the reputation of a node. The first part, J_{app}, is computed based on the requirements of the tasks/jobs that are going to be processed such as the resource type (32 or 64 bits), the OS (Windows, Linux, AIX, Mac), and the data required. For consistency, from now on, we define that a **job** is composed of many **tasks**. In Hadoop, there are two types of tasks: map and reduce. Within a job, all map tasks are the same, and so are all reduce tasks. The second part, J_{sys}, is computed based on the current system status such as CPU usage, network activities, free memory percentage. The third part, H_{app}, is the reputation of the node from the application aspect or the usual client (scheduler) point of view. Apparently, H_{app} is calculated based on the feedback of the usual client (scheduler) over a long time. The last part, H_{sys}, denotes the long-term system characteristics such as the reliability or availability of the node.

J_{app} and J_{sys} can be categorized as Just-in-Time reputation (JR) of a node, which describes its current status or recent behaviors within a small time window. H_{app} and H_{sys} are history-based reputation used to capture the past behaviors of a node. This classification arises from the observation that sometimes a reputation of a node can be affected temporarily. For example, if node A has a good reputation so far, the scheduler very likely assigns the task to it. Unfortunately, as we prefer to schedule tasks to A, it starts getting a heavier workload and, hence, processes the tasks very slowly. In fact, the reputation of that node at that particular time should be low so that our reputation-based scheduler doesn't assign tasks to it. Another observation is from the data locality point of view. Despite the fact that node A has had a high reputation so far, if it doesn't host the data required by the task, the scheduler should not choose it. In general, there are two dimensions of this JR. One relates to the task requirements, and the other relates to the current status of the nodes. To show the *openness* of Opera, we implemented JR (J_{app} and J_{sys}) as the predefined components and history-based reputation (H_{app} and H_{sys}) as user-defined components. The next step is how to quantitatively calculate them.

Let vector $R =< J_{app}, J_{sys}, H_{app}, H_{sys} >$ be the reputation of a node within $[t_0, t]$ and n be the number of node.

As mentioned above, J_{app} is computed from the job description. There are many requirements in a job description. Some of them are single-value, and the others are multi-value. For example, while the required OS belongs to the single-value type, the required data blocks belong to the multi-value type. Let x_i be the indicator variable of a single-value requirement and the target node be the node we want to calculate the reputation for.

$$x_i = \begin{cases} 1 & \text{if the target node fits the requirement } i^{th} \\ 0 & otherwise. \end{cases}$$

In certain circumstances, it is critical to meet multiple requirements simultaneously. For example, if a job requires running on a 64-bit system, we can't assign it to the 32-bit. If that's the case we compute

$$p_{critical} = \prod_{i=1}^{\text{The total critical requirements}} x_i$$

Note that $p_{critical}$ is equal to 1 only if all critical requirements are met; otherwise, its value is 0.

Another important multi-value requirement is the data required by the task. If a node has the data the task requires, it should have a high reputation toward that specific task. We capture this as

$$p_{data} = \frac{\text{The \# of blocks stored on the target node}}{\text{The total \# of required blocks of the task}}$$

At the end, the first part, J_{app}, is computed as the following

$$J_{app} = p_{critical} \times p_{data}$$

The second part, J_{sys}, demonstrates the current system status of a node. By current status, we mean in the short time $[t'_0, t'_1]$ which is much smaller than the window size $[t_0, t]$ in the computation of H_{sys} and H_{app}. In Ganglia, for example, these short periods are 5, 10 and 15 minutes. Basically, J_{sys} is computed from Ganglia's metrics. Since there are many metrics available in Ganglia, we can derive many ways to compute J_{sys}. One simple example, which takes into account both CPU and memory, is

$$J_{sys} = \text{the idle CPU percentage} \times \text{free memory}$$

If the CPU of a node is too busy and has less free memory, it should have a low reputation. It is worth noting that Opera can easily integrate other methods of computing J_{sys}, but that is beyond the scope of this work.

The third part, H_{app}, is the history-based reputation from the user point of view. We utilized the information in the log files of Hadoop to calculate it as followings.

$$H_{app} = 1 - \frac{\text{total number of failed/killed tasks}}{\text{total assigned tasks}}$$

From the scheduler point of view, a node that has more failed/killed tasks over its total given tasks should have a lower reputation. As initializing, Opera reads this information from the history directory of Hadoop and calculates the start-up reputation of all nodes. Then, after finishing each job, it reads the Hadoop log file and updates this type of reputation accordingly. This type of reputation is *augmented* after each job execution.

The last part, H_{sys}, is a purely system-related reputation. We can use either the availability or reliability of a node for this value.

$$\begin{aligned} H_{sys} &= availability(t) = \frac{\text{actual up time}}{t - t_0} \\ &= \frac{\text{total replies}}{\text{total heartbeats sent}} \end{aligned}$$

For the reliability, let $f(t)$ be the unconditional failure rate of a node

$f(t)$ = the probability that a node will fail between t and $t + d_t$ ($f(t)$ is a probability density function.)

Let F(t) be the cumulative probability that a node has failed by the time t

$$F(t) = \int_{t_0}^{t} f(t)dt$$

The cumulative probability that the node has not failed from t_0 to t, given the fact that the node was up at t_0, is the reliability of that node during $[t_0, t]$

$$H_{sys} = reliability(t) = 1 - F(t)$$

From this formula, if we know the distribution of the failure rate function, we can easily compute the reliability of a node. With the assumption of *memoryless* failures, we may apply another simpler formula:

$$H_{sys} = reliability(t) = e^{-(\frac{t}{MTTF})}$$

in which MTTF is the mean time for the node to fail.

4.5.2 The Selection Criteria

We implemented three selection criteria “above average”, “top one-third” and “mixed”. The first two were implemented as part of Opera representing predefined criteria. The last one was implemented in Hadoop as user-defined criteria.

The “above average” criterion return nodes that has the reputation greater than or equal to the average reputation of available nodes. The reputation here refers to a specific dimension of the reputation vector.

The “top one-third” criterion first ranks all candidates according to a specific dimension of the reputation vector and chooses nodes among the top one-third of the candidate set.

While the first and second criteria consider only one element of the reputation vector, the third one takes two of them into account. Since this is the user-defined criterion, it depends on the specific situation.

4.5.3 Hadoop Modification

After implementing Opera, we need to modify Hadoop to use it. In this subsection, we first discuss the Hadoop scheduler briefly then explain how to change it. The modification in the speculative execution is explained last.

Hadoop has two main parts: the file system (HDFS) and the MapReduce framework [49]. In Hadoop, a MapReduce job contains many tasks of two types: map or reduce. Hadoop uses a master/slave model and a polling mechanism (pull model) to schedule tasks. Every time a worker (Tasktracker) has available execution slots, it contacts the master (Jobtracker) to ask for a task periodically. After performing certain checks such as `is-blacklisted`, the master finds a task to assign to it.

To use Opera in such a model, the idea is that when a node comes to ask for tasks, the master asks Opera for its reputation, evaluates it against a selected criteria, and assigns tasks to it accordingly. To implement this, we developed an additional method called `shouldAssignTasks(tasktra` which is invoked when a tasktracker comes to ask for tasks, and it returns true if the reputation of the tasktracker meets the criteria. Actually, we can either specify new customized criteria right in this method or choose the predefined ones from Opera.

It is noteworthy that if we apply certain selection criterion to a fixed list of workers, we may decrease the system utilization. For example, among 10 workers, if our criterion is "choosing above median reputation nodes", the scheduler can never assign tasks to more than 5 workers. This is because we also include the nodes which already received tasks in the candidates list to assign new tasks. We should instead assign tasks to available candidates only. As a result, we maintained a list of available workers called `candidatesList` and applied the selected criteria to this list only. Before executing any job, `candidatesList` contains all workers in the system. In executing a job, it shrinks as the number of nodes that receive tasks increases, and grows as they finish tasks. Since the list keeps changing over time, we can always select nodes successfully.

In our implementation, Opera is also used in the speculative execution of Hadoop in which slow progress tasks are speculatively executed on other nodes. We used Opera to find such nodes. Basically, the new chosen nodes should be "better" (i.e has higher reputation) than the current hosts of the slow progress tasks.

4.5.4 Scalability and Availability

For internal system information, we employed Ganglia, which has been widely used and can handle a cluster of 2000 nodes. Consequently, we have scalability in collecting system information. For user-provide information such as feedback, since most rating messages from raters are often small in size, Opera can handle many of them easily. As a result, it is safe to claim Opera is *scalable*.

To increase the *availability* of the system, we can have a secondary Opera server as a backup plan. This secondary server periodically backs up the reputation vector from the ordinary server. In case of failure, since the history-based reputation isn't affected much in missing short-period information, we can still use the "out-of-date" historical reputation. For the JR, since Ganglia multicasts monitor information to a channel, the secondary server can easily obtain information from it and compute the JR themselves; hence, it always has up-to-date JR. In addition, Ganglia also allows the user to define their own metric; therefore, it is extensible and fits best to our Opera design described in the previous section.

4.6 Performance Evaluation

4.6.1 Experiment Setup

Since Opera is used to aid resource selection based on multi-property criteria, we need to create a heterogeneous systems to evaluate it. However, our available testbed is homogeneous in term of hardware because as most local institute clusters, it was built at the same time with the same hardware configuration. As a result, we created two types of heterogeneity, which are *availability* and *workload*, instead of heterogeneity in the hardware. Some nodes in the system are configured to fail during certain periods to provide the differences in availability. Since we focus on fail-stop only, we simulated a failure by disabling the network connection to that node (turn off the network interface) during predefined periods. The differences in workload were created by scripts that recompile the newest Linux kernel at a certain time to keep the CPUs busy and the memory full, and "ping" the local host heavily to pollute the network. As our previous assumptions, we configured the experiments so that they run only on half of the available resources.

Our testbed cluster has a total of 23 machines including 2 servers and is described in detail in Table 4.2. One server is the namenode of Hadoop. The others contain both the Opera server and JobTracker. It also runs Ganglia meta data daemon -*gmetad*- to collect information from monitor daemons - gmond. All other nodes are tasktrackers/datanodes and installed *gmond* (agents that collect information). The number of replicas in Hadoop is 3. The monitoring metrics are collected, sent and updated at different rates. For example, the CPU-related metrics are collected every 20 seconds by gmond at each worker. They are sent to the gmetad every 90 seconds, and the gmetad updates its own RRD database every 15 seconds. In addition, these rates are different between metrics.

The applications we used are packaged together with the Hadoop distribution. They are all in Hadoop's example package and are MapReduce programs. We utilized three programs: *Sort*, *Grep* and *WordCount*. Their functionalities are self-described by their name. The Grep and WordCount input is a 7.7 GB text file generated from log files and a Shakepeare play. Grep finds a given pattern in a given input file. The search pattern is written as a regular expression, and the matched results are stored in an output file. WordCount counts the number of occurrences of each word in a given input set. The input of Sort is 3.9GB and is generated randomly by the *Randomwriter* - another built-in application of Hadoop.

Table 4.2: The Opera's testbed.

Type	**Machines #**	**CPU**	**Memory**	**OS**
Server	1	4x2GHz	4GB	RHEL5 x86_64
Server	1	1x2.34GHz	1GB	Fedora 8
Workstation	21	2x1GHz	512MB	Ubuntu 8.04

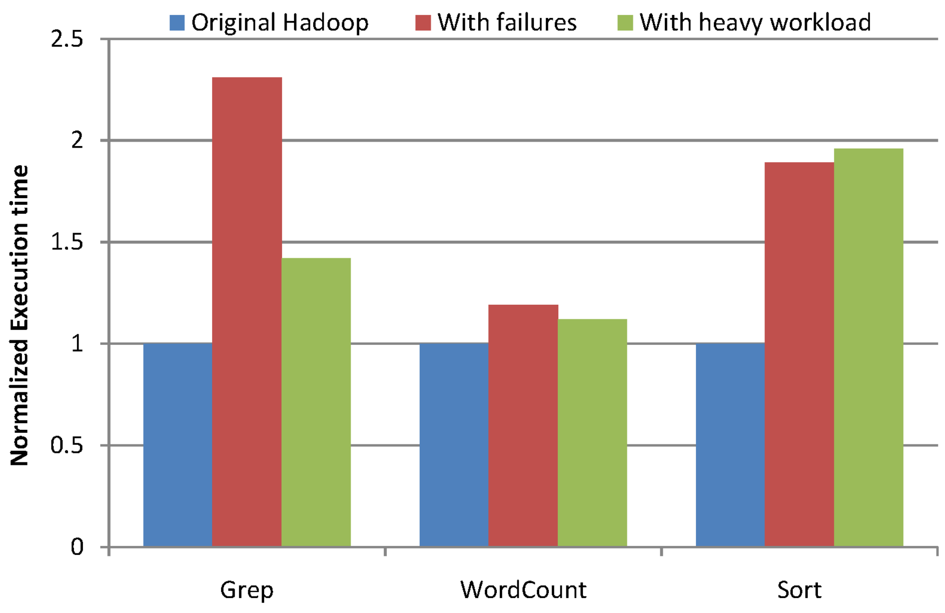


Figure 4.3: The effects of heterogeneity on Hadoop performance.

4.6.2 Metrics

We used two main metrics in this evaluation: *execution time* and *the number of failed/killed tasks* of a job. The execution time, as usual, represents the performance of the system. The number of failed/killed tasks expresses the usage efficiency of the system. Intuitively, the higher this value, the lower the efficiency is because we spend resources such as energy, computation, network bandwidth on executing these failed/killed tasks but receive nothing. We should calculate the usage efficiency explicitly here, but since its computation requires information about the resource (CPU time or energy) each failed/killed task has spent before it fails or is killed, we don't have it at this moment.

4.7 Heterogeneity Analysis

We first show that our two heterogeneous types, *availability* and *workload*, also affect Hadoop performance as Matei's work on [139] although we used different methods. Figures 4.3 and 4.4 show the execution time and the total number of failed/killed tasks of three applications in Hadoop under failures and other workloads. The number of failed nodes and high workload nodes are nearly half of the system (8/21 and 9/21, respectively). In these experiments, we normalize the measurement to the healthy system which has speculative execution enabled yet has no failures or heavy workload.

The failure periods are chosen so that the whole job is still successful because as we extend these periods, the number of failed task attempts increases and so does the job execution time. At a certain level, if we keep extending them, the whole job fails as the number of task failures of any task reaches its limitation, which is 4 by default.

In the systems with failures, the executed time increases 19%, 131% and 89%, and the number of failed/killed tasks increases 94.73%, 77.78% and 267% for *wordcount, grep,* and *sort*, respectively. In the systems with heavy workloads, the executed time increases 12%, 42% and 96%, and the waste increases 5.26%, 44.45% and 103% for *wordcount, grep, sort*, respectively. With these defects caused by heterogeneity, the next subsection will detail the

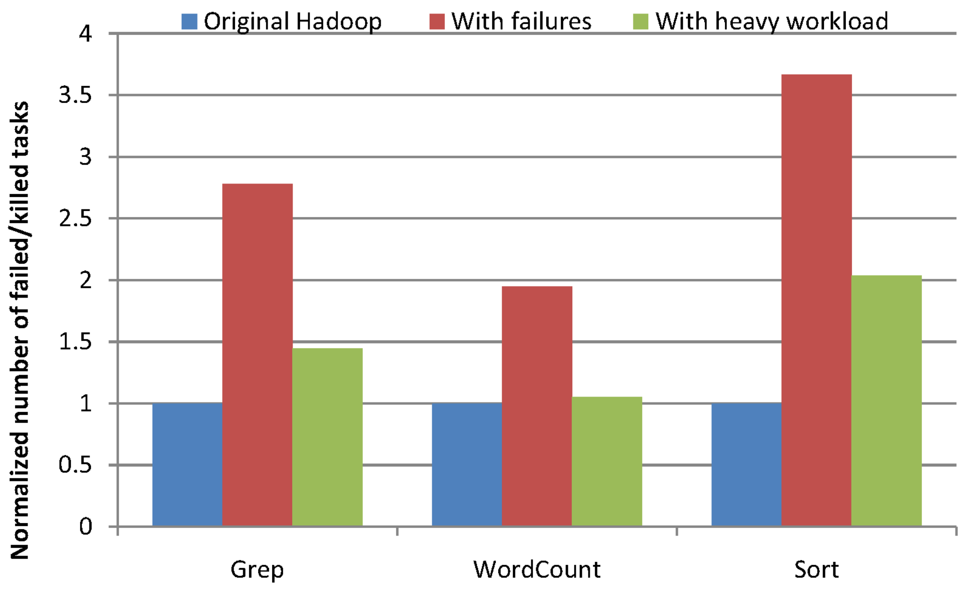


Figure 4.4: The effects of heterogeneity on Hadoop number of failed/killed tasks.

improvement by Opera.

Table 4.3: The results of using different availability.

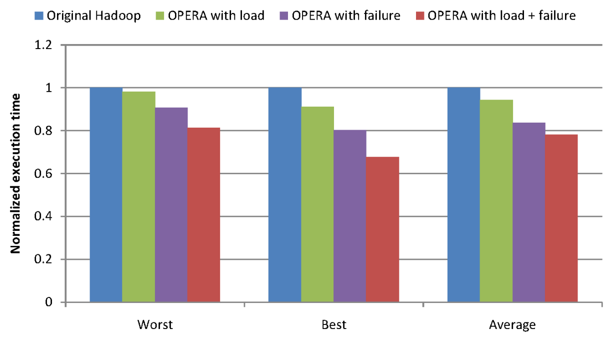
	Original Hadoop			Opera		
	worst	**best**	**average**	**worst**	**best**	**average**
Execution time (s)	706	652	678	538	507	523
Number of failed/killed tasks	27	17	23	16	12	13

4.8 Improvements

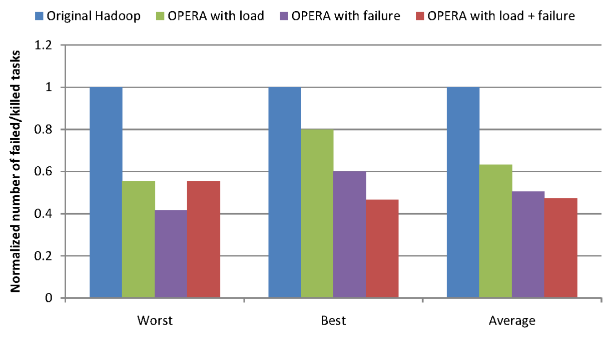
Our goal in this section is to illustrate that with the help of Opera, the scheduler can avoid nodes with either low availability or high workload, or both. The Sort benchmark is going to be used as the primary workload from now on because it is popular and also used as the main benchmark for evaluating Hadoop at Yahoo [139].

Table 4.4: The results of using different workloads.

	Original Hadoop			Opera		
	worst	**best**	**average**	**worst**	**best**	**average**
Execution time (s)	662	546	606	563	499	539
Number of failed/killed tasks	22	11	18	9	5	7



(a) Execution time



(b) The number of failed/killed tasks

Figure 4.5: A comparison of three different ways of using two heterogeneous types.

There are two factors that affect the selection process and, hence, are considered here: *the reputation element* and *the selection criteria*. While the reputation element factor represents which dimension(s) we consider, the criteria is the policy that is applied on it(them). For instance, the selection policy "choosing the nodes that have J_{sys} above the average" means the "above average" criterion is applied to the element, J_{sys}, of the reputation vector. As a result, to study the effects of each of these two factors, we run the experiments with different values for one while keeping the other fixed. The following two subsections describe in detail the effect of the reputation elements, which in fact represent many different aspects of the heterogeneity and the selection criteria. The last subsection is comparing Opera with the LATE scheduling [139].

4.8.1 The Heterogeneity

Each aspect of the heterogeneity is captured by an element in the reputation vector. In the previous section, we mentioned two types of heterogeneity: availability (failures) and load. Essentially, they are the two aspects of the heterogeneity that we consider and are captured by H_{sys} and J_{sys}, respectively. In this part, we are going to do the experiments with each single aspect of the heterogeneity first and then with both. The selection criterion factor applied in these experiments is fixed. It is the "above average". Tables 4.3 and 4.4 contain the results of running Sort with and without Opera on a system with heterogeneity in availability and load, respectively. Since the results vary greatly among executions, the tables list the worst, best and average values of the results to give an idea of their range.

From Table 4.3, one can see that the execution time and the number of failed/killed tasks are reduced by 23% and 41% on average, respectively, when using Opera under the presence of failure. Similarly, under the presence of other workload, the execution time and the number of failed/killed tasks can be improved on average by 11% and 59%, respectively, as shown in Table 4.4.

Finally, we did the experiments with four configurations as depicted in Figure 4.5 under an environment that has **both** failure and heavyloaded nodes. It is worth noting that this environment differs from that of the two previous experiments. From the figure, we can see that the last configuration, which took into account both availability and workload, outperformed the others. Its performance was improved 26% in comparison to the original Hadoop. It demonstrates the ability to handle criteria built on more than one element of the reputation vector. This makes Opera unique, separating it from other previous work.

It is worth to note that tasks of a job in Hadoop fail independently which means that only the failed tasks need to be re-executed. If Opera is used in the models in which task failures are highly correlated and require the whole job to be re-executed, the efficiency improvement would be much more significant.

4.8.2 The Selection Criteria

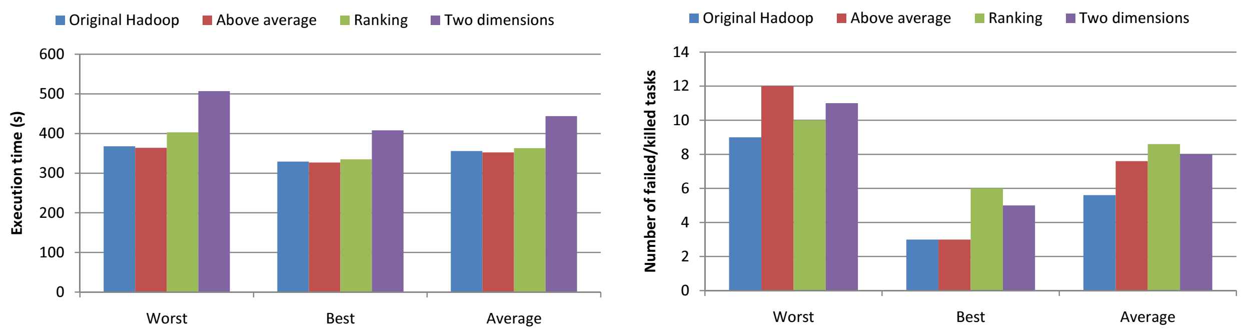
Now we are in a position to study the effects of the selection criteria. We run the Sort again on Hadoop with different selection criteria yet under the same, *ideal environment* (no failures, no other workload). Obviously, in such an environment, Opera brings no benefits but overhead since it is designed for heterogeneous environments. Consequently, with this experiment design, we cannot study only the effects of different selection criteria but also the overhead of Opera.

The Sort was run at least 5 times for each configuration. There are a total of 4 configurations. The first one is the original Hadoop. The rest three use the three criteria defined in

Section 4.5 to select nodes.

Intuitively, one can tell that the execution time increases as the criteria become more complicated. Moreover, in the ideal environment, Opera should expose its overhead over the original Hadoop. In other words, the execution time should increase from the first to the fourth configuration. Figure 4.6 consolidates this intuition. On average, the execution times of the first 3 configurations are almost the same while the last one introduces a 25% additional delay. The reason is that the last (use-defined) criterion is more complicated than the other two. Furthermore, the selection code is executed every time a worker asks for tasks which is quite often. The number of killed/failed tasks varries greatly among executions. For example, the worst case in the original Hadoop is 9, and the best one is 3. On average, Opera has about 2 extra killed/failed tasks.

One may wonder why the performance of the two dimension criteria is worse in this part than the others while it is better in the previous part. This is because the previous experiments were done in a hostile environment which contains both failures and other load. To reduce this overhead, one possible approach is to use feedback control mechanism to select suitable criteria since we have a lot of monitoring information.



Execution time Number of failed/killed tasks

Figure 4.6: Study of three different selection criteria on Opera.

4.8.3 Opera and LATE

As mentioned earlier in Section 4.2, LATE [139] also improve the performance of Hadoop in heterogeneous environments. Although this is not exactly the same goal with us, our approaches are somewhat related. Therefore, we would like to compare the performance of Opera and LATE here.

There are four configurations in this experiment. For each of them, we ran the sort application on the same input of 1GB of data three times. The first two configurations are LATE with and without failure. Since LATE has been integrated into Hadoop version 0.21.0, we used that version as LATE configuration in the experiment. The remaining two configurations are Opera with and without failure. The criterion we applied here for Opera is of the type "above average", and there was no other workload in the system.

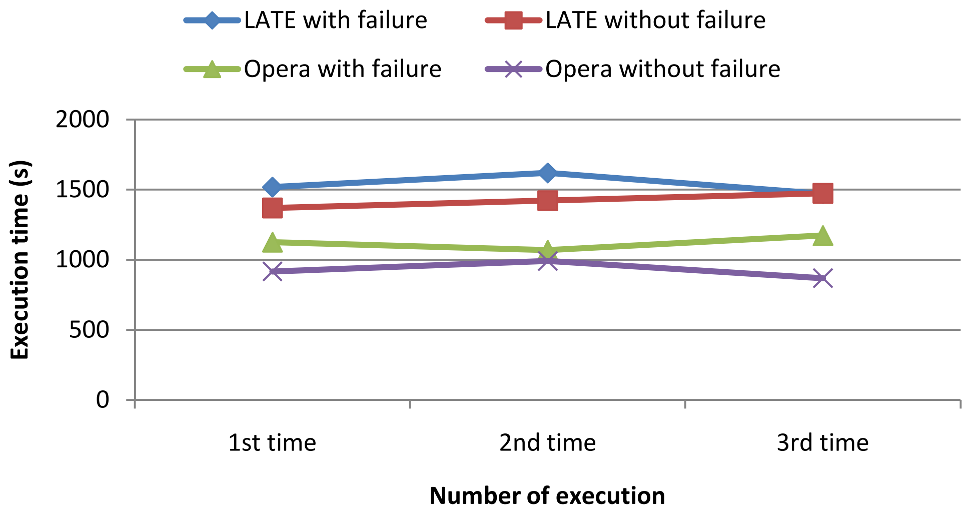


Figure 4.7: The performance of Opera and LATE.

Figure 4.7 shows the results of the experiment. In the figure, we can observe the outperform of Opera over LATE. This improvement can be explained as the result of the difference in the time of scheduling and the way to select the tasktrackers. While Opera is performed right at the beginning of the job execution, for every tasks, LATE focuses on the speculative tasks and is only trigger when stragglers are detected. While Opera tries to select "good" candidates for every task assignment, LATE focuses on which tasks should be speculative executed but it does not care who will take care of such tasks.

4.9 Effects of Sampling rate

In this section, since we used a monitoring tool polling the clients to capture the behavior of the system, we would like to study the effects of changing the sampling rates to the performance of the system. However, we only focus on changing rate in the JR because it doesn't make much sense in the history-based part. Basically, the history-based reputations are used to predict the future based on the past information, which is expressed in traces such as failure trace. The past information is often measured during a long period of time, which is much longer than the execution time of a job. The sampling intervals in this situation are often long; therefore , during the execution of a job, only a few data are sampled. These data clearly cannot affect the history-based reputation severely.

Figure 4.8 shows the results of sorting 10.5GB of data under the same heavy load distribution with different sampling rates. Intuitively, the higher the rates, the more accurate the monitor is but the higher traffic in the network. As we can see, when we increase the sampling interval, the monitor loses the sensitivity of capturing the behavior of the system and, hence, produces worse performance. Further, we notice that increasing the sampling rate too high is not good because the benefit in performance improvement cannot outweigh the network overhead at some point. For example, in Figure 4.8 the performances at the sampling intervals of 20 and 30 seconds are essentially the same.

4.10 Discussion

Although we implemented the calculation of all 4 elements reputation vector, the above evaluation only used 2 of them (J_{sys} and H_{sys}). We didn't use J_{app}, which represents the job requirements and data locality, because of two reasons corresponding to its two constituents. The first one is related to the critical requirements. It is trivial to avoid nodes that don't meet

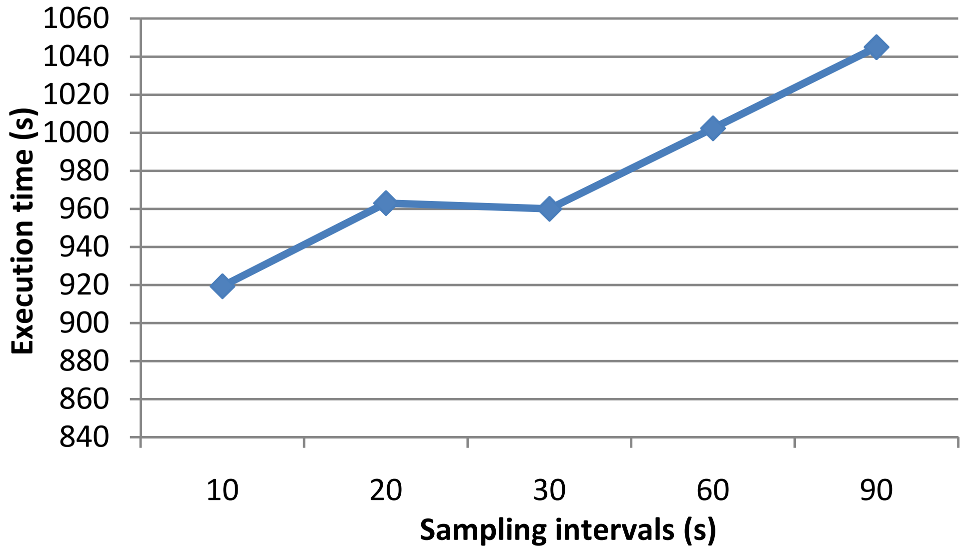


Figure 4.8: The performance of Opera with different sampling rates.

them because these nodes all have the reputation of 0. It make no sense if we put, for example, some Windows or Mac machines to the systems, set the job description files, and show that Opera doesn't choose them. The second reason relates to the data locality. We ignored this too because Hadoop claims that it already took into account the data locality as it schedules map tasks.

For H_{app}, actually, it represents the traditional reputation systems in which the customer is the Hadoop's master node and the service providers are the workers. After each transaction (job) the customer (master) rates the service providers (workers). We introduce H_{app} to demonstrate that Opera can easily cover the traditional reputation. We skipped H_{app} because ,unlike the commercial systems, our specific system doesn't assume the malicious behavior of nodes. All service providers are objective and try to behave correctly. Therefore, even if a node has had a high number of killed/failed tasks, we shouldn't punish it *again* because they were done unintentionally. We used "again" because the objective behaviors were already considered in other dimensions.

In these experiments, we assumed that the future behavior of nodes can be obtained exactly from their reputation vectors. In reality, this assumption is too strong. We can only predict such information with a certain probability. Fortunately, the research about such prediction is well studied [92, 89, 111, 96], and the node behavior estimation, especially in the near future [141], is actually accurate enough.

4.11 Energy Efficiency Evaluation

In this section, we evaluate the (energy) usage efficacy of using Opera. In this experiment, we used 4 wattsup meters [17] to measure the power dissipation in the whole system. This time, for the sake of simplicity, we only used and measured 20 slave machines. We did not measure the servers since they mostly did the same job for all configurations. The actual computation happens in the slaves, and the data is also stored in and transferred between them too. Three wattsup devices were used to measure the power of 4 machines each. The last wattsup measured the rest 8 machines.

The power dissipation was measured at the finest granularity available of wattsup devices which is 1 second. At this granularity, the reading software sometimes could not read info from the devices. There are several missing values and, therefore, we only show here the average of the remaining power values over the interested time periods. The time of all machines in the system during the experiment was guaranteed to be synchronized.

Since the wattsup devices kept recording data to files during the experiment and the time is synchronized, we only need to mark the periods of interest to compute the average power in such periods later. For example, to measure the base line power (no applications but the OS), after starting all machines in the system and waiting for them to be stabilized, we recorded the time start, wait 5 minutes and recorded the time end of this period. Then after the whole experiment finished, we compute the average power during this recorded period for each wattsup and add them up to have the average power of the whole system.

In this experiment, we measured the power of the whole system while there is no application except the OS (base line); running only Hadoop; and running Hadoop with three MapReduce applications: grep, wordcount and sort (with different configurations each). The base line power of the cluster is 1711.66 Watts and the power of running Hadoop (without any job execution) is 1718.59 Watts. We can see that running Hadoop itself raises the average power to 7 Watts in comparison to the base line.

There are three configurations for each application. This first one is running the applications with the original Hadoop without any failure. The second one is executing them with failure using original Hadoop. We did not include heavy load here because we only want to measure the energy of our job. Using our current measurement method, we can't exclude energy consumption of such load. The third configuration is using Opera with failure.

For the Hadoop with application configurations, we run the application several times, compute the average power as well as the execution time of each configuration. From these values, we can derive the average energy consumption of the whole system for each configuration and the usage efficiency (defined in Section 4.1). These results are shown in Figure 4.9 and Table 4.5.

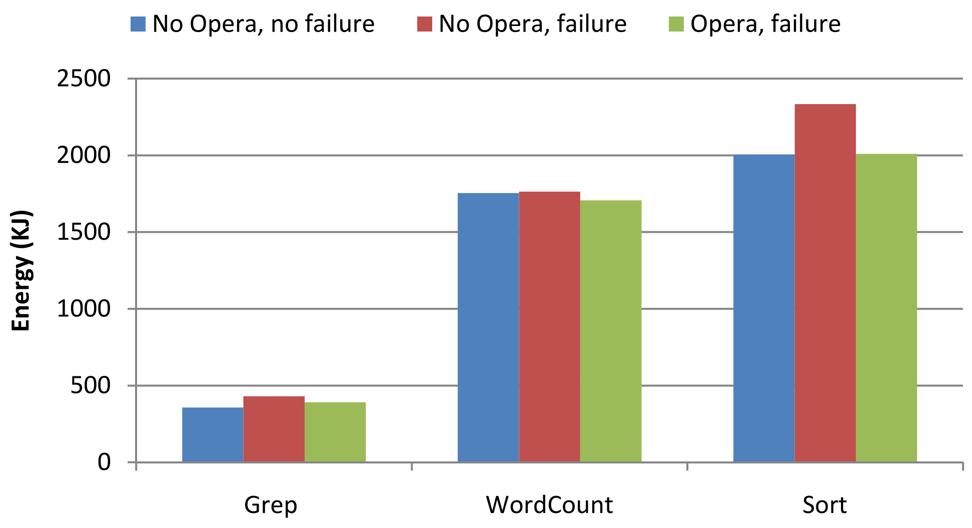


Figure 4.9: Energy consumption of Opera in three applications.

From Figure 4.9, we can observe that running applications in Hadoop increased the energy consumption significantly (in comparison to the base line). The figure also points out that, with failure, the energy consumption of the original Hadoop is increased in all cases, but it

Table 4.5: Comparison of energy usage efficiency in terms of three applications.

	Grep	WordCount	Sort
No Opera, no failure	100%	100%	100%
No Opera, with failure	83%	99.43%	85.90%
Opera, with failure	91.20%	102.76%	99.79%
Improve over the original Hadoop	9.87%	3.35%	16.17%

is decreased when employing Opera. The improvement is not significant in the Grep and WordCount applications because such applications has only one reduce, and the reduce phase dominates the execution time of the whole job. While the benefit of Opera comes from selecting good candidates, this selection only happens once in the reduce phase of such applications and therefore, they are not benefit much from Opera unless the original Hadoop selects a "bad" node for the reduce task.

Table 4.5 details the usage efficiency of the whole system with different applications and configurations. In general, Opera is always more efficient than the original Hadoop with failure. It's worth to note that although the usage efficiency of the WordCount almost does not hurt from failure (they have one time-consumed reduce task), it is still improved by Opera. Opera can even be more efficient than the ideal case (no failure) thank to the significant reduction in the number of failed/killed tasks.

4.12 Summary

To improve the efficacy in using resources, such as energy, we need to reduce the resource waste. Particularly, in this section, we would like to lower the number of task re-execution, by choosing appropriate machines to execute the tasks. We successfully designed and developed Opera, which is used to select machines from a candidate set in the systems based on their reputations. The system not only allows us to reduce the waste caused by failure but also enables users to select service providers by predefined as well as user-defined selection criteria and enables reputation researchers to develop their own reputation computation models.

In this chapter, we proposed using vectors to represent reputation and experiment results proved that it is a suitable approach to deal with the heterogeneity and energy efficiency. We set up a specific scenario which is using Opera to aid Hadoop scheduler to select workers. Based on the scenario, we developed four reputation computation models. Two of these models are built-in and the others are defined outside Opera to show the extendibility of Opera. The experiment results expressed both the benefits and cost of using Opera.

CHAPTER 5

DIFFERENTIATED REPLICATION

By having the Opera service in place (as described in the previous chapter), this chapter will employ it to offer differentiated services on top of a heterogeneous system.

Our targeted system is expected to provide differentiated services to the users since it is the foundation of Cloud infrastructure. There are many users and each of them has different requirements to the system. This chapter presents our attempt in this area from the data storage view point. We are going to augment the existing file system APIs to enable difference in availability of the data. The work on this chapter in the system is shown in figure 5.1.

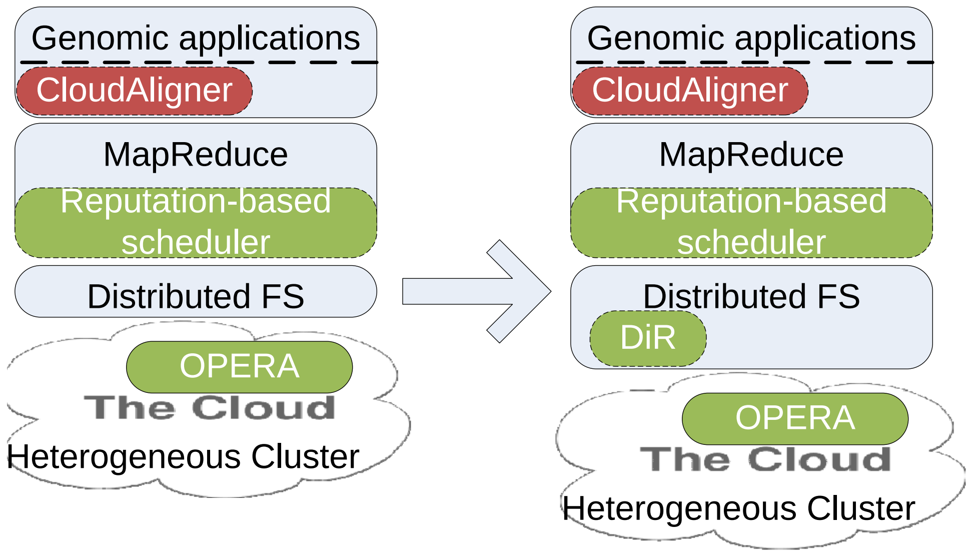


Figure 5.1: System with Differentiated Replication Service (DiR).

5.1 Introduction

It is clear that many users in cloud computing have different requirements for a service. Some require the best performance, while others may ask for the reliability of their data to be a priority. In general, the requests may demand different properties from services such as availability, reliability, durability, and performance. There may be one, several, or all of these properties under consideration for a particular request. This possibility demands differential services. However, most available cloud services today do not take this fact into account. They often only provide one type of service for all users. For example, Windows Azure and Amazon S3 maintain a fix number of replicas (3) for each data stored in it. Both Microsoft Windows Azure and Amazon S3 guarantee in their service level agreement (SLA) that the availability of customer's data is always greater than 99.9%. This may lead to poor resource utilization from the provider or an inefficient usage from the user. We also find that all of these properties relate to replication. Differentiable replication strategies can provide different availabilities, reliability, durabilities, and performances. Therefore, we propose a new strategy called *Differentiated Replication (DiR)* to address this problem.

To demonstrate the concept of DiR, we built a prototype system providing storage services that are capable of providing different availabilities. Users are provided a simple interface that allows them to *store* and later *fetch* their data with their expectation. The current version of DiR provides four replication types.

The contributions in this section are threefold. First, we propose the idea of differentiated services for data centers, and design a set of simple but powerful APIs for high level users. Second, we propose **four** different replication strategies on the server side, enabling differentiated services in terms of availability. Third, we implement a prototype of DiR, and evaluate the proposed four replication strategies comprehensively in terms of availability using both synthetic and real failures traces. The evaluation results show that the last replication strategy, which takes both user requirements and system behavior into consideration, indeed is capable of providing different availabilities and execution times.

5.2 System Design

5.2.1 Assumptions and Requirements

The goal of DiR is to build a read/write only (not modified) storage system that provides different types of replication services to the user with better resource utilization. We target read/write only storage system for the sake of simplicity in term of data consistency. However, this is also practical because the data in the Cloud is often very huge and should not be modified. This assumption is often made in the area of data intensive computing like HDFS of Hadoop [8].

We mainly focus on differences in: (1) replication strategies; (2) search algorithms; (3) network topology; and (4) availability. Hence, in our case *better service* may mean higher availability or faster or less communication cost. The idea for a user to request other properties such as durability, reliability, and performance is almost similar and will be introduced later. The system is heterogeneous with different hardware, software, computational ability, etc. Each member can join or leave the system or fail at any time. Given the requirement of read/write only service, the system does not need to maintain consistency between replicas. Therefore, the only two methods we need to provide are *store* and *fetch*. In addition, the failure we consider is of a fail-stop type rather than the Byzantine failure type. This means that when the machines are alive, they are supposed to have correct behavior. Finally, reasonable load balance, fault-tolerance, scalability and reliability are also important requirements of the system.

5.2.2 APIs

Users of our system are not terminal application users but developers of front-end applications. They are not supposed to know replication techniques in detail. The interface component is required to be simple enough so that it can easily be used in applications. Therefore, we only need to add one more parameter, called *service-type*, to the current APIs of Chord/DHash [129] to indicates which type of service the user requires. These methods merely call new methods with a default service-type, as illustrated in Table 5.1. The user will notice that the service-type of a *fetch* needs to match that of the *store* for a certain file.

Function name	Description
`fetch(filename)`	retrieve a file from the DiR system
`fetch(filename,service-type)`	retrieve a file from the DiR system with specific service type
`store(filename)`	insert a file from local file system to DiR
`store(filename, service-type)`	insert a file from local file system to DiR with specified service type

Table 5.1: The DiR APIs.

5.2.3 Availability Analysis

Basically, there are two ways to provide different levels of availability: change the number of replicas, or change the location of them. Higher availability of an object can be achieved by increasing the number of its replicas or by placing its replicas onto more "available" machines. Intuitively, more replicas on more reliable nodes will produce higher availability. Even so, we cannot tell which method provides better availability in some situations. The system has to decide the availability under the resource constraints to guarantee the load balance. If we do not handle this correctly, we may introduce extra overhead to highly available nodes. The problem is formalized as follows.

Given an expected availability A of a certain object/file, and a set of nodes with their own availability, we need to find the number of replicas, and the specific nodes in which to store them. It is noteworthy that the availability of an object stored on a machine is equal to the availability of that machine, also under the fail-stop assumption.

Let M be a set of nodeIDs and corresponding availabilities of N nodes.

$$M = \{(n_i, a_i) | n_i \text{ is the ID of node i and } 1 \leq i \leq N\}$$

Let

$$\sigma = \left\{ \{(n_{x_l}, a_{x_l})\}_{l=\overline{1,k}} \;\middle|\; \begin{array}{l} (n_{x_l}, a_{x_l}) \in M, \\ n_{x_i} \neq n_{x_j}, 1 \leq i, j \leq k, \\ 1 \leq x_i, x_j, k \leq N \end{array} \right\}$$

The solution of the problem is in σ set.

Assuming $\{(n_{y_1}, a_{y_1}), (n_{y_2}, a_{y_2}), \ldots, (n_{y_l}, a_{y_l})\}$ is one specific solution, the following approximation should be satisfied

$$\begin{aligned} A &\approx 1 - (1 - a_{y_1})(1 - a_{y_2}) \ldots (1 - a_{y_l}) \\ &= f(a_{y_1}, a_{y_2}, \cdots, a_{y_l}). \end{aligned} \tag{5.1}$$

Note that in Equation (5.1) while A represents the user's expectation, $\{a_{y_1}, a_{y_2}, \cdots, a_{y_l}\}$ represents the availability of the system. With a certain value of A, we may have several solutions.

One way to completely solve this is: For every member m_i of σ, compute $A' = f(m_i)$, if $A' \approx A$ then m_i is a solution. Unfortunately, this method is $O(2^N)$, in which N is the number of nodes.

The second method is to calculate the average of all the availabilities

$$\overline{a} = \sum_{i=1}^{N} a_i$$

and use the formular in [31]:

$$l = \frac{log(1 - A)}{log(1 - \overline{a})} \tag{5.2}$$

to derive the number of replicas l. This approach may create a resource utilization problem in the next step of choosing proper nodes to store replicas.

The third method is shown in the following algorithm

This method may cause an overload in the high availability nodes.

Finally, since this problem is of a constraint programming type, the efficient way to solve it is to use an existing C(L)P solver.

As a result, no matter what method we use, from the system design point of view, the system is required to have a monitor server to provide the availabilities of all nodes in the system. This

Algorithm 1 Availability Computation.

```
1: Sort $M$ in decending order of availability
2: $F \Leftarrow (1 - M[0].a)$
3: $i \Leftarrow 1$
4: while $(i < N) and (A' < A)$ do
5:     $F \Leftarrow F * (1 - M[i].a)$
6:     $A' \Leftarrow 1 - F$
7:     $inc(i)$
8: end while
```

leads to the need for *OPERA*.

5.2.4 *OPERA*

Opera stands for **OPE**n **R**eput**A**tion model, which is a general model to compute the reputation of nodes in the system. *Opera* allows users to define how to calculate reputation, and it returns the reputation of nodes based on that definition. It employs a traditional master-slave model in its communication, since we need to obtain the global reputation of the system. *Opera* clients communicate to each other in replying to the rate request from the server. Design and implementation of *Opera* will be detailed later in Section **??**. Basically, Opera employs Ganglia (a monitoring tool) to collect information about nodes in the system and calculates reputation for each node based on this information.

5.2.5 Utilization Analysis

We argued that DiR also provides better resource utilization. This is rather obvious and straightforward. Let's define

$$\text{the utilization of a system} = \frac{Res_{real}}{Res_{need}}$$

in which, Res_{real} are the resources that really used to provide the services and Res_{need} are the resources that can satisfy user needs.

The following analysis compares the resource utilization of DiR and that of uniform replication, which is a very widely used technique today. We assume both systems have n requests

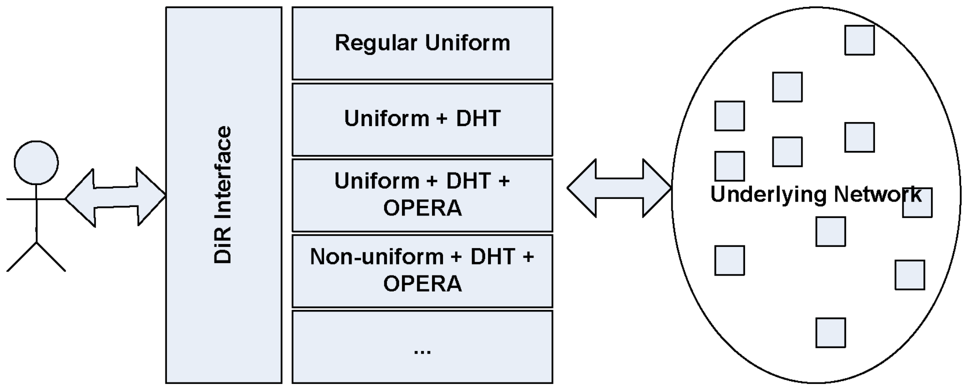


Figure 5.2: The DiR system architecture.

$r_1, r_2, \cdots, r_n$ and c_i is the correspondent number of replicas to satisfy r_i. The total number of replicas of DiR (idealy) and uniform method is $\sum_{i=1}^{n} c_i$ and $\sum_{i=1}^{n} Max\{c_i|0 \leq i \leq n\}$ respectively. Note that, in the uniform repication system, we need to choose l large enough to satisfy the highest quality requests. For example, with $n = 4, c_1 = 2, c_2 = 3, c_3 = 2, c_4 = 4$, the resource utilization of ideal DiR and uniform system is 1 and $\frac{4\times4}{2+3+2+4} = 1.45$. As a result, this analysis proved that DiR used resources more efficiently. From another aspect, with the same resource, DiR (better utilized system) can satisfy more requests as well.

5.2.6 DiR System Architecture

The overall system architecture is shown in Figure 5.2. The user uses the DiR Interface to ask for service. Depending on the request, the DiR interface decides which replication strategy to use. There are, in total, four replication strategies available (represented by four blocks in the figure) that can offer all required differences. In fact, there are many other options to choose to construct a strategy. For example, we can choose random walk search algorithm [99], CAN [110] or Pastry [115] to build a new type of service. Such openness is expressed by the lowest block with “three dots” in the figure. Finally, the rightmost circle with small squares inside represents the physical underlying network connecting the machines of the system.

Our system can also support the differences in the durability/reliability of objects stored in it by modifying the policy being used in the monitoring server, *Opera*, to define how to calculate them. The nodes that are more durable/reliable have higher reputation scores. By doing this, the *Opera* server returns the durability/reliability of all nodes in the system. The remaining problem is how to calculate these values.

5.3 Implementation

We implemented our system by extending Chord/DHash [129]. The architecture of an individual node (peer) derived from our previous design is shown in Figure 5.3. The rightmost block is the Opera client. This block is in charge of rating other nodes and responding to the request from them as well as from the Opera server. The highest level in Figure 5.3 is the DiR interface that offers the APIs to the user and chooses an appropriate handler. The two leftmost blocks (File Transfer and breadth-first search) correspond to the "Regular Uniform" handler of Figure 5.2. "File Transfer" is used to receive files sent from other nodes. "Breadth First Search" is used to find a replica. The last three middle blocks (DiR Manager, DHash and Chord) correspond to the final three handlers in Figure 5.2. The lowest block, Chord, is a replica lookup service. The upper block that uses the Chord lookup service is DHash, a block store service. This layer is responsible for storing/retrieving blocks of data into/out of storage devices. The DiR Manager is located on top of the block store layer (DHash). It is used to provide the `store/fetch` file functions to the DiR interface, communicate to Opera and calculate the appropriate number of replicas as well as their locations.

We have implemented a prototype of a DiR storage system and Opera using C++ on Linux. The DiR prototype has implemented all parts of the previous design. It offers several types of differences: in the network topology (unstructured or structured), communication model (message and aRPC), search algorithm, replica location and expected availability. These differences are implied in the following four service types (corresponding to the four handlers in the Section 5.2.6) and are summarized in Table 5.2.

Type S_r indicates that a user wants to use a normal uniform replication with a traditional search algorithm. Nodes communicate with each other by sending and receiving messages. Al-

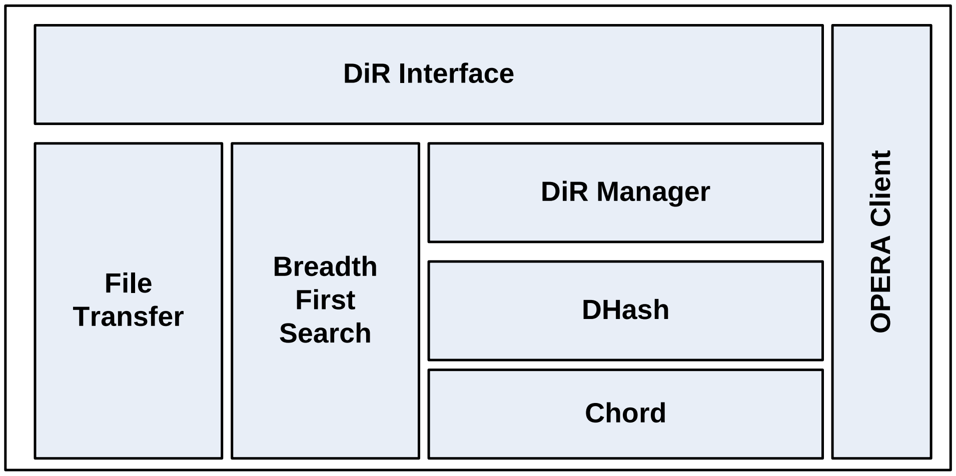


Figure 5.3: An augmented node in DiR implementation.

Table 5.2: The summary of available service types

Service type	Description
S_r	Regular uniform replication, unstructured network
S_d	Uniform replication, DHT, ring-based
S_{d+o}	Uniform replication, DHT, ring-based, Opera
S_{d+o+a}	Non-uniform replication, DHT, ring-based, Opera

though this strategy is popular, we built it from scratch since we could not find any open source implementation available online. To handle this type, each node has two servers. The first one, (called searchserver), waits to receive a file request and looks for the file in the system. This server is the "Breadth First Search" block in Figure 5.3. To make the search process workable, we apply the TTL (Time To Live) technique to each request. The second server, (called file-transferserver), is used to receive actual files sent from searchservers. It is worth noting that since we use failure traces in the experiment, these servers have to be implemented to tolerate failure at anytime. Therefore, we also apply a timeout technique in the communication.

Type S_d represents the uniform replication, using Chord as a DHT lookup algorithm. In fact, this type is the original version of Chord/DHash.

Type S_{d+o} utilizes the same lookup algorithm as S_r but with a different replica location. With this service type, DiR first calls Chord to get the successor nodes of the hash value of

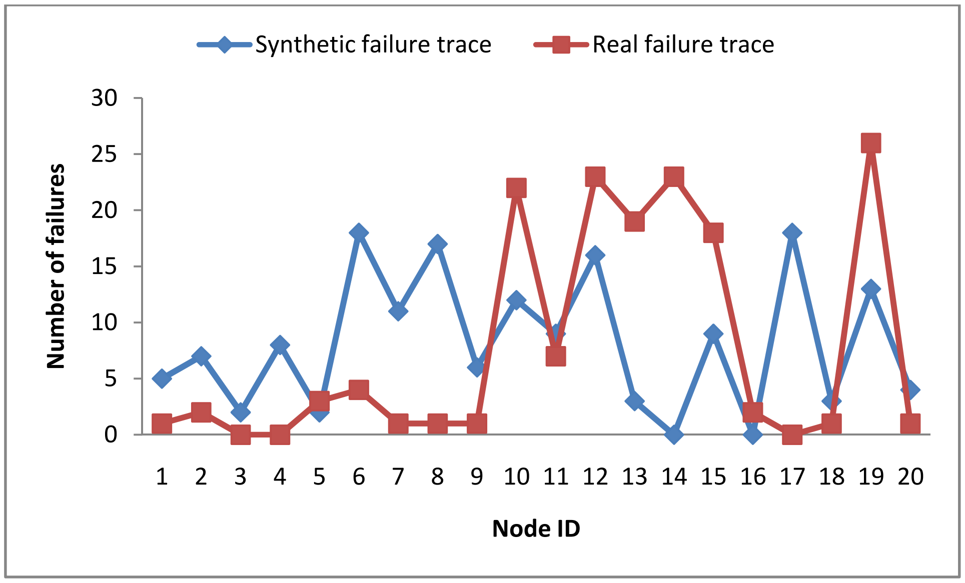


Figure 5.4: Number of failures.

the filename and contacts Opera to ask for their availabilities. Based on this information and the predefined number of replicas, DiR chooses nodes with the highest availability to host the replicas of that file. The thing worth being noticed here is that in this prototype, these first three types have **the same number of replicas**.

Type S_{d+o+u} means the user only cares about the availability, not the number of replicas. This version of DiR **fixes the target availability to 3 "nines"** as in Amazon S3 storage service, but this can easily be changed to the desired values. DiR calculates the necessary replicas using equation (5.2) with the average availability of the successor list of the file ID. In the step of choosing host machines to store the replica, we simply choose the sets randomly, recalculate the availability accordingly of each set, and choose the one that is close to the expected availability. Although this way may not provide the best solution, it is faster and helps to balance the load.

5.4 Experiment and Results

To evaluate our system, we deployed DiR onto a cluster of 21 nodes. The first 20 nodes had DiR installed. The 21st node was dedicated to the Opera server with a guarantee not to fail during the experiment.

To prepare for the experiment, first, we need to have a files generator and distributor to create and distribute files randomly to 20 nodes. The total number of files in the synthetic trace is 400, and that of the DZero trace [51] is 25,951 files.

Second, for request and failure traces, we used both synthetic traces and modified real traces. Figure 5.4 shows the detailed number of failures in both synthetic and real failure trace. The total failure duration time of each node is displayed in Figure 5.5. The simulation time for the synthetic traces is 50 minutes. The real failure trace is from the availability information of the first 20 nodes of Microsoft PCs trace [35] measured during 35 days since July 6, 1999; and the real request trace is from a physical application of the DZero experiment [51] on April 2004. Since it was not practical to conduct the experiment for an entire month, both real failure and request traces are scaled down to one day only. In scaling down the request trace, we encountered the problem of congestion. This is because the request trace was forced to request too many files at the same time. As a result, we modified the trace so that the time to ask for the file is slightly different if they are the same in the original trace. The availability results

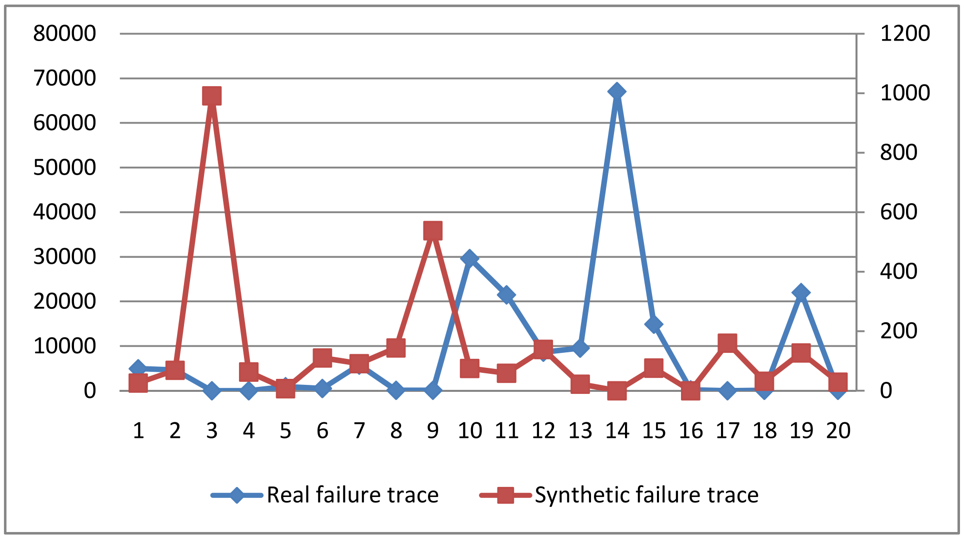


Figure 5.5: Failure duration time in the failure trace.

returning from the Opera server are shown in Figure 5.6.

In this section, we measure two main metrics: *execution time* and *availability* of different

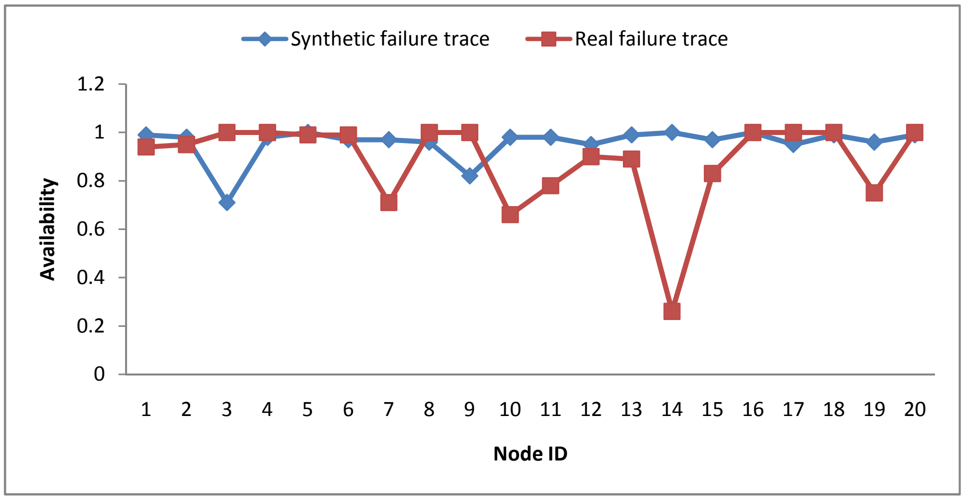


Figure 5.6: The availability of the system.

Table 5.3: The availability of DiR under synthetic and real traces.

Service-type	$S_r(2)$	$S_r(3)$	S_d	S_{d+o}	S_{d+o+u}
Availability (synthetic trace)	44.62%	86.40%	61.62%	70.06%	98.13%
Availability (real trace)	28.02%	29.32%	82.17%	89.79%	99.95%

service types.

Table 5.3 shows the availability of the system using the synthetic traces and the modified real traces. From the synthetic results, we can see significant improvement in the availability of the S_r with different configurations of the neighbor list (two or three neighbors). In addition, we also found that the availability of S_{d+o} is better than that of S_d. This means we can improve the availability of Chord/DHash with the aid of Opera. The table also shows that we cannot tell which type is "better" generally. One can argue that $S_r(3)$ (with 3 neighbors) is the best. However, this statement is true for the availability only. It is easy to see that, the $S_r(3)$ strategy costs more resources (in term of link number and bandwidth) than the others. From the real results, the availability of S_{d+o} is also better than that of S_d. The availabilities of S_r are poor because the real request trace requests several hundred files at the same time; and together with the failure trace, it crashed some of our search servers and hence, produced those poor availabilities.

Another experiment was about S_{d+o+u} alone. Figure 5.7 shows the availability of 10 different files of random size. The horizontal line in the figure represents the expected availability that was set to three nines by default. The result was measured in 50 minutes.

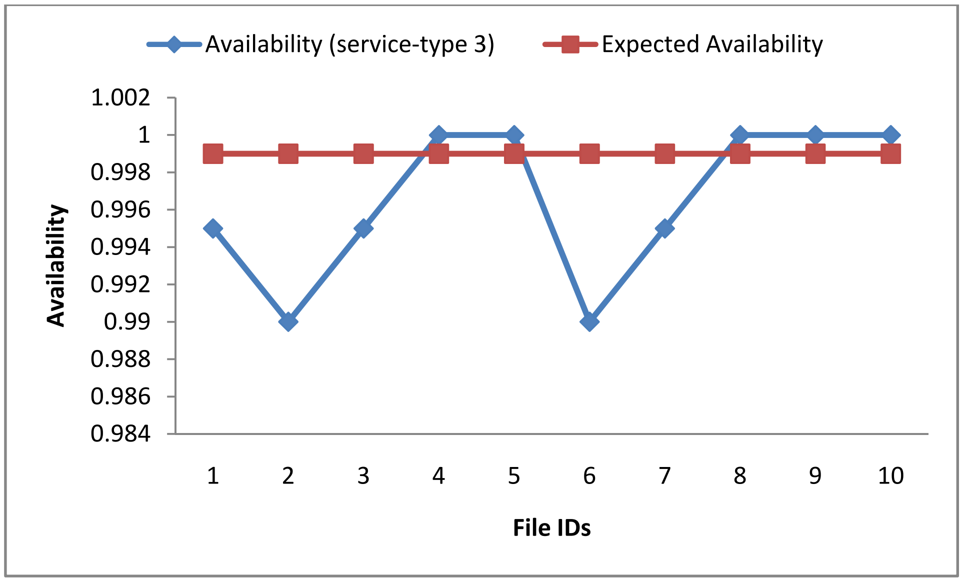


Figure 5.7: Availability of files using S_{d+o+u}.

To measure the performance of DiR, we created files with various sizes, inserted them into the system and then retrieved them. We only measured the response time of the successful requests. Assuming that there was no failure, we got the results shown in Figure 5.8.

5.5 Related Work

Differential service is one of the key aspects of DiR, encompassing flexible availability, reliability, durability, file placement, and search methods. Many studies are related to and have led to the culmination of DiR. Beside Chord/DHash [129], perhaps the closest work to ours is the Total Recall [31]. Total Recall file system of Bhagwan *et al.* also provides an option to choose the availability of objects. However, their users are in fact system administrators. They aimed at relieving administrators' burden by maintaining the degree of replicas automatically. Another paper of Zhong *et al.* [142], which also employed the non-uniform replication, considers the optimal number of replicas of an object for high availability to be directly pro-

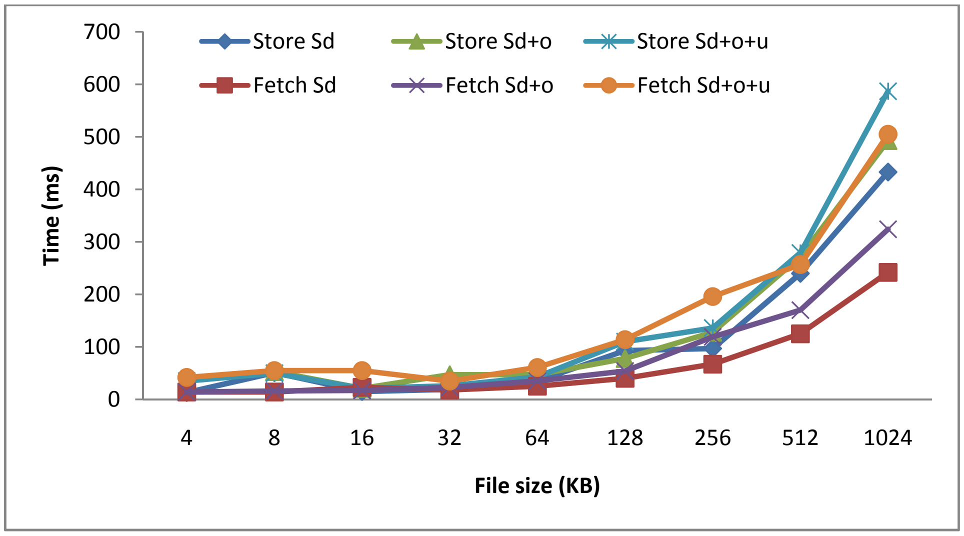


Figure 5.8: DiR performance of S_d, S_{d+o} and S_{d+o+u}.

portional to the object's popularity. However, their approach does not consider the different type of services from users. Also related to replication strategy, Cohen [45] finds the optimal replication strategy, in term of search size, lies between a uniform and a proportional strategy.

Different placement algorithms were introduced in [39, 40, 52, 91]. To increase availability, Giwon's work in [105] is concerned with dynamically replicating objects, and Acunam's group uses a fixed number of replicas depending on peer availability [59]. In regard to durability, Wang *et al.* focuses on the durability of data through replication [132], while Chun's research improves the durability of large amounts of data using a replication algorithm [43]. Besides the breadth first search and the Chord mentioned in the previous sections, [99] presented a search algorithm using multiple random walks to improve performance over the Gnutella like flooding search method.

5.6 Summary

Diversity in the requirement of services will soon be an important feature and requirement in cloud computing. In this chapter, we took the first step to address this problem. Focusing on the replication technique, we proposed the concept of differentiated replication that can offer different types of services, and developed a prototype storage system focusing on the availability.

CHAPTER 6

INTERNAL DATA MOVEMENT

At this point, from the previous chapters, we know that Cloud computing is a beneficial solution to deal with large data in genomics. Also, some system challenges, particularly in terms of heterogeneity, in building a system to support data management and analysis in genomics such as heterogeneity and energy efficiency were identified and tamed. In this chapter and the next one, we turn our focus to another fundamental challenge: the movement and processing of big data. Many people may think that there is no new problem with data and that the algorithm is more important. However, dealing with extremely large data is completely a different story. Such large scale of data creates challenges that are unforeseen. For example, the traditional popular file systems such as FAT32 or ext3,4 are failed to manage such data. If we use the distributed file systems, the issues such as consistency and availability have to be considered. It is worth to note that the solutions discussed in this chapter can also be applied to any research fields that have big data. It is not necessarily limited to genomics only.

Since, in practice, the data is often generated in a place that is different from where it is processed and analyzed [21], the frontier scientists who decided to adopt the Cloud and Mapreduce solutions often conduct the following steps to obtain the analytic result of the data:

- The data is moved to the data center
- The analysis software is executed against the data inside the data center or the cloud. For example, with our context (similar to the Amazon EMR), the user first prepares the input data and the execution files which is the analysis software written in the MapReduce framework. Then he or she creates and launches a jobflow that contains all the information needed to execute their job such as the number of instances, instance types, application jar, parameters, etc. Behind the scene, after receiving the jobflow, the sys-

tem instantiates the requested instances using the special Machine Images that contain a MapReduce implementation and starts the MapReduce framework system automatically. It then copies all data from the persistent storage such as S3 to the computing nodes and executes the application. The results of the execution are written back to the persistent storage because after finishing the jobflow, the system terminates all instances.

- The result in the persistent storage in the cloud is downloaded to the local machine of the scientists for record or further analysis.

From this workflow, we can categorize the data movement in this popular scenario into two different types: internal and external. Internal data movement refers to data transmission among machines inside the data center. External data movement is the data transmission from/to user to/from data center. This chapter will discuss about the internal data movement and the next chapter targets the external one.

As usual, Figure 6.1 spots the relative position of the work in this chapter in the whole system. In the figure, the DLA VM scheduling includes a Data location aware VM scheduler and a distribute cache.

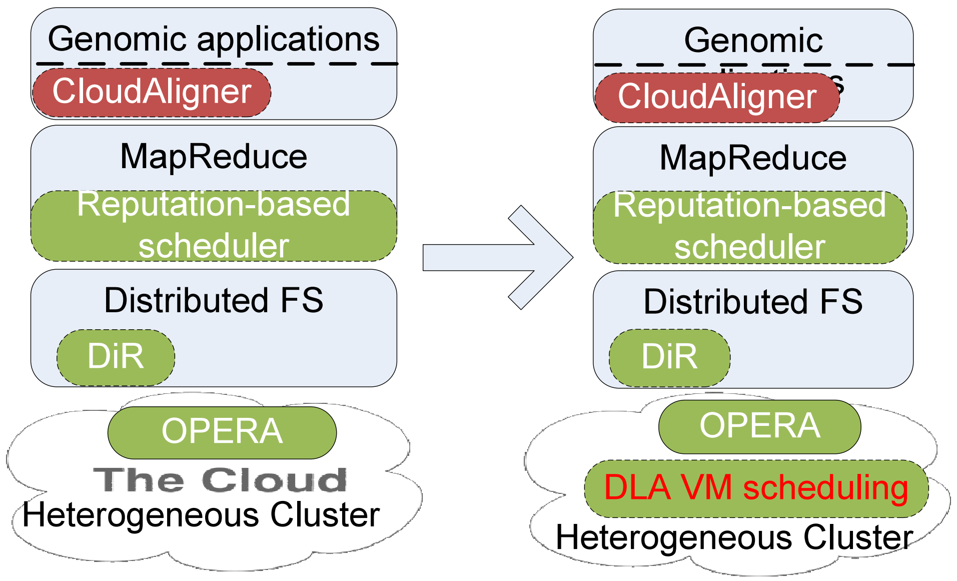


Figure 6.1: Internal Big Data Movement.

We have witnessed the fast growing deployment of Hadoop, an open-source implementation of the MapReduce programming model, for the purpose of data-intensive computing in the cloud. However, Hadoop MapReduce hasn't been originally designed to run transient jobs, in which users need to move data back and forth between storage and computing facilities. As a result, it is not efficient (wasting a lot of resources) when operating in the Cloud. In this paper, we first show the inefficiency of the existing MapReduce system in the Cloud, then study its causes, followed by a solution. The system we studied is Elastic MapReduce by Amazon — a popular implementation of the MapReduce system in the Cloud. Two main issues leading to the inefficiency were identified. The first is the extra overhead of using virtual machines. The second is in the data movement process. The MapReduce framework is proposed for data intensive computing processing very large amount of data. Transferring such data to computing nodes is very time-consuming and also violates the rationale of Hadoop which is moving computation to data. However, it happens frequently in the current implementation of Elastic MapReduce. To address the data movement problem, we developed a distributed cache system and a virtual machine scheduler. Testing our prototype on a private Cloud, we found that the performance of the system can be improved significantly when running different applications such as CloudBurst, CloudAligner, Sort and Grep under the warm cache.

6.1 Introduction

The recent significant increasing the volume of data in in physics, biology, lifesciences, simulation, data mining, etc requires an efficient model of computation, and Google's MapReduce [49] is a popular model, because it is suitable for data-intensive applications such as web access log stats, inverted index construction, document clustering, machine learning, and statistical machine translation. There are several implementations of MapReduce such as Phoenix [135], Sector/Sphere from UC [63], Hadoop [8] (from Yahoo) and Mars [65]. One of them, Hadoop, is so popular that Amazon – one of the Cloud provider leaders – offers a separate service called Elastic MapReduce (EMR) based on Hadoop. In the last few years, we have witnessed the fast growing deployment of Hadoop, an open-source implementation of the MapReduce programming model, for the purpose of data-intensive computing in the cloud. Motivated by the popularity of Amazon's Elastic MapReduce system, we choose EMR as our target system for detailed analysis since it is the most success commercialized example of running Hadoop in the Cloud.

Through our detailed analysis in Section 2, we found that Hadoop is not as efficient as we expected when running in the Cloud, because of two reasons. The first obvious drawback is the virtual machine (VM) overheads which include the JVM since Hadoop was developed with Java. A Hadoop MapReduce job is typically executed on top of JVM operated inside another virtual machine if running in the Cloud. Our experiment results show that the execution time of the same application on VMs is about 4 times longer than that on physical machines. The second issue is related to the extra overhead caused by data movement between storage and computing facilities when running in the Cloud. In Elastic MapReduce, data has to be moved online to the Hadoop VM cluster from the Amazon's simple storage services (i.e., S3) transiently. As Hadoop is best known for its capability in data-intensive computing in which the amount of process data is often extremely large. Such temporary data movement is a great burden on the infrastructure, which in turn wastes a great deal of resources such as network bandwidth, energy, and disk I/Os. For example, when sorting 1GB of data on our testbed, the data movement time is 4.8 times larger than the sorting time. In this work, we focus on the data movement problem and leave reducing VM overhead for future work.

We designed and implemented a distributed cache system and a VM scheduler to reduce this costly data movement process. Under the warm cache scenario, which is usually true once the cloud starts running for a while, our system can achieve performance improvement by up to 56.4% with two MapReduce-based applications in lifesciences, 75.1% with the traditional Sort application and 83.7% with the Grep application.

The rest of the chapter is organized as follows. In Section 6.2, we will first show the inefficiency of EMR in terms of performance as well as data access to motivate our work. Then Section 6.3 and 7.3 will describe the design and the prototype implementation of our distributed cache system to improve the data movement of EMR. The performance evaluation of our prototype will be shown in Section 6.5 before discussing related work in Section 6.6. Finally, the conclusion will be drawn in Section 6.7.

6.2 Problem Statement

In this section, we will first explore how current system (Cloud and EMR) works before identifying its potential issues. In our case, the first question would be *How do Cloud providers generally store user data?* and the next one is *How does EMR work with respect to data processing?*

6.2.1 Background

In this chapter, we focus on Infrastructure as a Service (IaaS) system such as the Amazon Cloud and Eucalyptus [104]. Since the insight of Amazon Cloud is very limited, we have to use Eucalyptus, an open-source Cloud computing platform that has a similar interface with Amazon EC2 (computing service) and S3 (storage service). While we know that Eucalyptus supports both ATA over Ethernet (AoE) and SCSI over Infiniband (iSCSI) storage networking standards, the architecture of Amazon S3 hasn't been published. However, through our work, whenever possible, we will run experiments on Amazon.

For storage service, in Eucalyptus, Walrus is a storage service interface compatible with Amazon's S3 allowing users to store persistent data, organized as buckets and objects. The interface of the block storage service in Eucalyptus is also compatible with Amazon's Elastic Block Store (EBS). As a result, EC2 commands can also be used to interact with it. With Amazon, each EBS volume is stored on two separate storage arrays for failure tolerance with low cost. Amazon EBS is stored externally on dedicated storage boxes. It is connected to EC2 instances via the network. In theory, EBS provides better performance and reliability in comparison to the ephemeral storage of the instance [5].

We believe that S3 is also in a storage area network separated from the EC2 because of two observations. First, the bandwidth between S3 and EC2 instances is smaller than that between EC2 instances[130]. Second, the separation between computation and storage in data centers is a common design. While S3 is designed particularly for storing persistent data with strict requirements on security, availability, reliability, scalability, etc, EC2 instances are usually used to store transient data which is destroyed during the instance termination if it is not moved to the EBS or S3. In the Eucalyptus Community Cloud, the volumes and buckets directories (similar to EBS and S3) are also located in the front-end node which is separate from the nodes that host VMs [4]. Further information about the differences between S3 and EBS can be found at [122].

To use EMR, the user first prepares the execution jar files (in the MapReduce framework) and input data, creates and launches a jobflow and finally obtains the results. The jobflow contains all the information needed to execute their job such as the number of instances, instance types, application jar, parameters, etc. Behind the scene, after receiving the jobflow, EMR instantiates the requested instances using the special Amazon Machine Images (AMI), that has Hadoop integrated into it and starts the Hadoop system automatically. It then copies all data such as the application jar and input from persistent storage such as S3 to the computing nodes and executes the application. The results of the execution are written back to the persistent storage since after finishing the jobflow, EMR terminates all instances.

6.2.2 Problem Identification

There are two issues with using EMR. First is the performance degradation from using VMs, and the other is data movement overhead. While this work concentrates on the data movement problem, the reducing VM overhead is left as the future work.

The overhead of using VMs instead of physical machines can be considered as a conventional problem. Much work has been done on this such as [69, 97, 70]. Most concentrate on the general platforms (Xen, KVM, VMware, etc) providing better solutions for virtualization. With the MapReduce framework, although the performance limitation of Hadoop on VMs was explored at [71, 72, 123], their comparison is not fair enough. They used two quad-core 2.33GHz Xeon processors, 8GB of memory for physical nodes to compare with 1 VCPU and 1 GB memory of the VMs. Thus, we did our own experiments to clarify that this is still an issue even with a fair comparison.

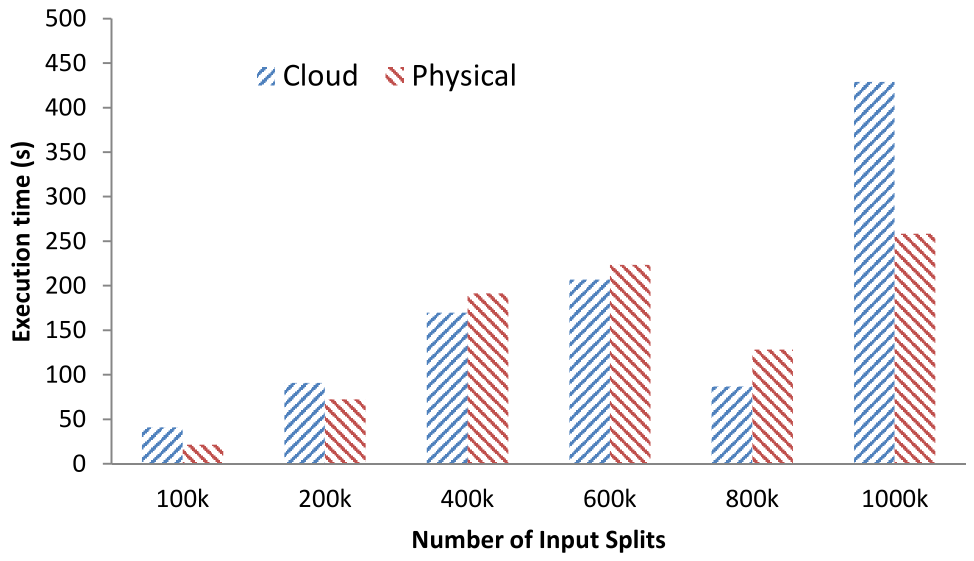


Figure 6.2: Data input transfer time on VM and Physical cluster.

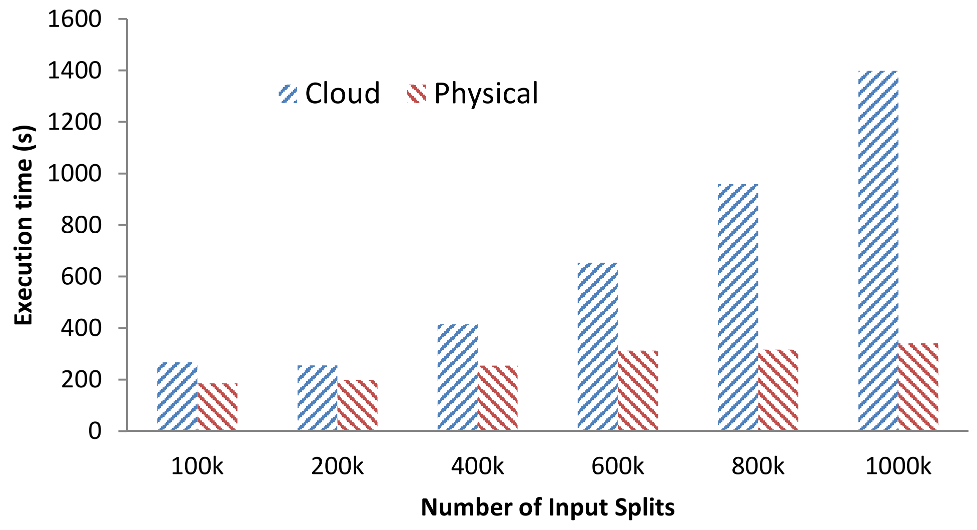


Figure 6.3: Execution time of CloudAligner on VM and Physical cluster.

Figures 6.2 and 6.3 show the time to load data and the execution time of CloudAligner [103], a fast and full-featured MapReduce based tool for sequence mapping in bioinformatics, in the VM and physical environment, respectively. In our experiments, the VMs and physical machines have the same configuration (CPU and memory). The detailed information about the input data of this experiment will be discussed later in Section 6.5.

In Figure 6.2, with small size data, running on VMs occasionally has even better performance. However, with large size data, running on physical machine is much better. For example, the data transferring time on physical cluster is only 60% of that on VMs with 1000k input splits. In Figure 6.3, as the size of the input data increases, the execution time overhead of running on a VM cluster also increases notably. For example, with 1000k input splits, the execution time on the VMs is about 4 times longer than that on the physical cluster. It is good to note that with this experiment, one physical machine only hosts one VM. Therefore, it's expected that the difference is clearer with multi-tenancy configuration as we can see in [139].

The second issue which is also the target of this work is the overhead of data movement. The above workflow of EMR implies that the user data has to be transferred between S3 and EC2 every time a user runs a jobflow. Although this transfer is free of charge from the users' point of view, it consumes resources such as energy and network bandwidth. This cost is, in fact, *considerable* since the MapReduce framework is often used for the data-intensive computing domain in which the data to process is often very huge. Therefore, the cost of moving them cannot be negligible.

To verify this observation, we carried out two experiments to show that the data movement portion is not a negligible part of the jobflow. Both were done on our private Cloud with Eucalyptus. The rationale and the correlation between our private Cloud and Amazon will be discussed in the next section. In the experiments, two different applications (CloudAligner and Sort) with varied workload sizes were executed to measure their execution and data movement time. Sort is a built-in benchmark of Hadoop. The results are shown in Figure 6.4 for CloudAligner and Figure 6.5 for Sort. It's easy to see that the data movement is the dominant

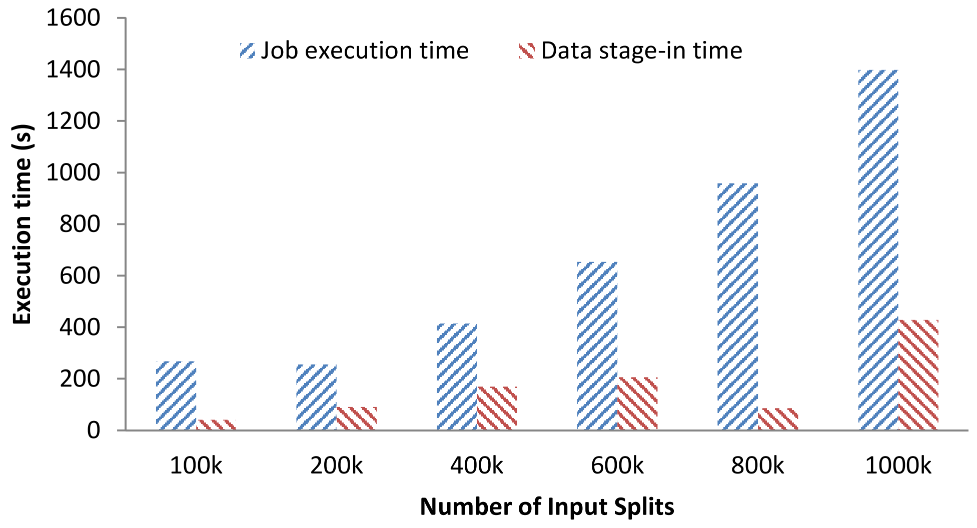


Figure 6.4: Data movement vs. Job Execution with CloudAligner.

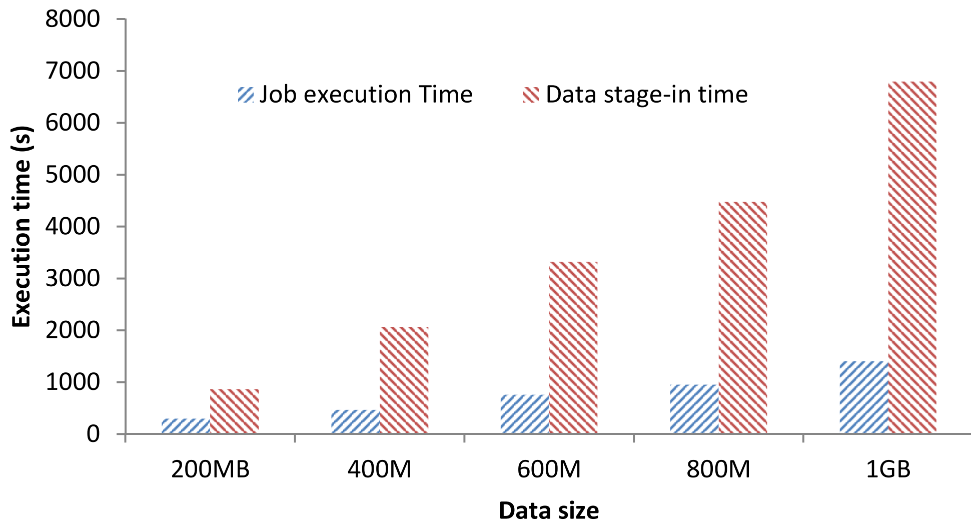


Figure 6.5: Data movement vs. Job Execution with Sort.

portion in the Sort application (4.8 times of the execution time) and is also an increasing portion in the CloudAligner application. It is worth noting that the largest input data in the experiment with CloudAligner, although extracted from the real data, is less than one tenth $(1/10)$ of the real data, and the selected reference Chromosome (chr22) also has the smallest size among others. This implies that the data movement portion in the CloudAligner experiment would also be significant with the real-size data. Therefore, it can be concluded that the data movement time tends to dominate the execution time in data intensive applications.

In general, the current approach of EMR violates the rationale behind the success of Hadoop, i.e. *moving computation to the data*. Although optimized to perform on data stored on S3, EMR still achieves better performance with input data on HDFS than on S3, and this is shown clearly in Figure 6.6. The figure contains the results of running CloudBurst on an EMR cluster of 11 small EC2 instances with different data size on data stored on HDFS and S3. CloudBurst is a MapReduce application used to map the DNA sequences to the reference genome [117]. Like CloudAligner, this type of application is a fundamental step in bioinformatics. It was selected because it can also be selected from the example application list of EMR.

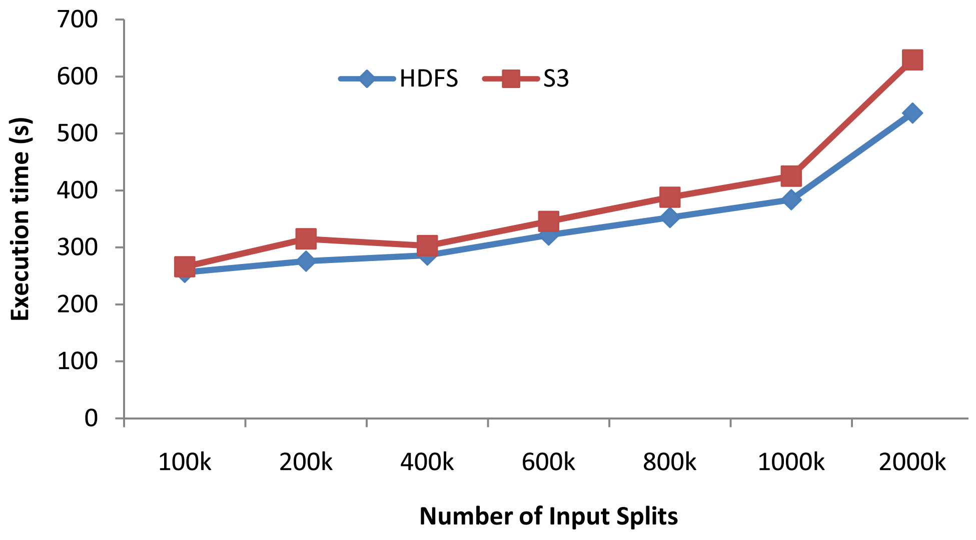


Figure 6.6: Execution time of CloudBurst on HDFS and S3.

6.2.3 Summary

Now we know that data movement is time-consuming, and, in EMR, such a process has to be repeated every time a user executes a jobflow because with the elasticity and transiency of the Cloud, such data is deleted after finishing the jobflow together with the VMs. The situation is even worse when, after the first run, the user simply wants to slightly tune the parameters and re-run the application on the same data set. The whole process of transferring the same amount of huge data has to be triggered again. One may argue that they can keep the original job alive and use the EMR command line tool to add the steps with modified parameters. However, based on our observation, for simplicity many users use only the web-based tool which is currently rather simple and has not supported such features. In addition, keeping the system alive also means the user keeps paying.

Therefore, this generates a great waste in bandwidth resource. To relieve this, the user persistent data and the computation instances have to be close to each other. We propose a system that caches the user data persistently in the physical hosts' storage on which the virtual machines of that user are hosted. This way, when the same user comes back to the system, the data is ready to be processed.

6.3 System Design

As concluded in the previous section, our ultimate goal is to improve the data movement of EMR by reducing the amount of data to be transferred to the computing system from the persistent storage. To achieve this goal, the loaded data should be kept at the computing instances *persistently* so that it can be used later and, when a user returns to the system, his or her VMs should be scheduled close to the data.

6.3.1 Terminologies, Assumptions and Requirements

Before going further into the details of our solution, first we will clearly identify terminologies, the target system we are considering, the aim applications, the requirements, etc.

In this chapter, we use virtual machine (VM), instance, node and computing node interchangeably. Further, physical machine and physical hosts are the same. The back-end storage is referred to as the persistent storage services such as S3 or Walrus. The front-end or ephemeral storage is the storage at the computing nodes. However, the front-end server denotes the head node of the Eucalyptus Cloud.

The specific target system into which we would like to add our cache should be similar to EMR because the EMR is a proprietary production system and therefore cannot be accessible.

The intended application of our system is of the same type as the target domain of the MapReduce framework, which is a data intensive application. Using our modified EMR system, the user would not see any change in the system except for the performance improvement. The interaction between the user and the system also remains unchanged. The user needs to create jobflow and specify instances, parameters, and executable files, etc; as usual.

The input assumption of our system, which is the same as EMR, is that the user data and their executions were already stored in the persistent storage (SAN) like S3. Therefore, handling data availability, reliability and durability are not necessary here. If our cache does not contain enough data (replicas) of a user, it can always be fetched again from the back-end storage.

In addition, like with Hadoop, the data we are targeting has the property "write one, read many" since the data is often extremely large in size, which creates much difficulty when modifying or editing it. Normally, the files in such a system are often read-only or append-only. Therefore, the strong consistency between the cached data and the back-end (S3) is not required here.

Since the user data is stored at the physical machines which host the VMs of different users, the system is required to ensure the security of these data. In other words, the cache needs to be isolated from the local hard drives allocated for VMs.

6.3.2 Solution

As we know, VMs and all the fetched data vanish after the jobflow finishes. Therefore, the general idea to reduce the wasted data movement is to cache these fetched data *persistently* in the computing cluster. The data should not be deleted together with the VM termination. We also know that the VMs need to be hosted on physical machines. Consequently, each physical host would reserve a portion of its local hard drive for this cache separated from the partitions for VMs. Another option here is to use EBS since it is also persistent.

Figure 6.7 shows how the users view the system. Each user has a set of VMs, and each VM has its own cache. Many VMs and caches of different users can share the same physical host.

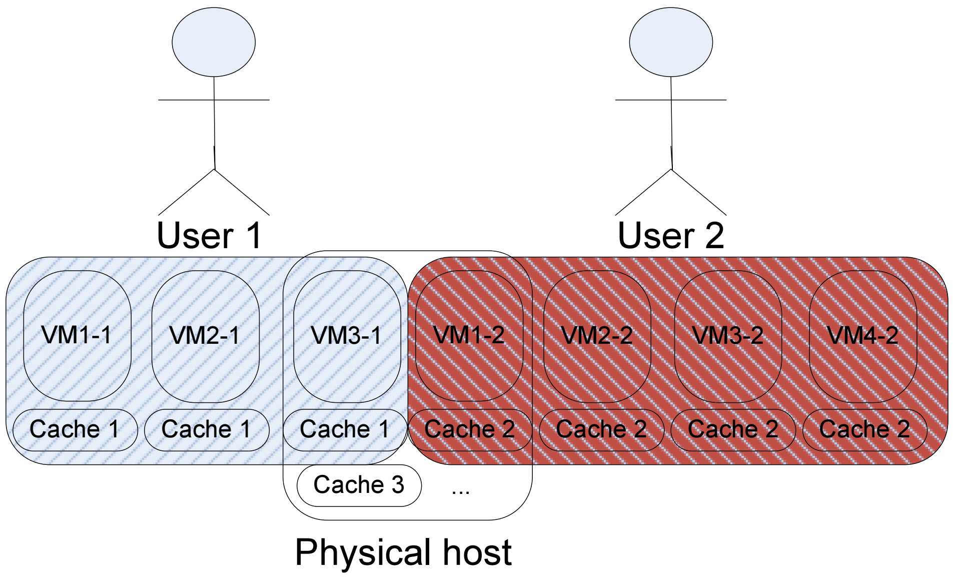


Figure 6.7: The system from user point of view.

However, the cluster is composed of many physical servers, and most Cloud users only have their VMs running on a very small part of it. This means the data of a certain user is only stored on a small number of physical servers of the cluster. There is no guarantee that the VMs of a certain user are always hosted on the same set of physical servers between two different jobflow executions. This results in the unavailability of the cached data to the user.

There are two possible solutions. We can move the missed fetched data to the machines that host the new VMs of the user. Or, we can modify the VM scheduler to host the VMs of a user on the same set of physical machines among different jobflows. Each has pros and cons. For the first option, although modification of the VM scheduler is not needed, we need to keep track of the location of the cached data and the new VM hosts of all users to identify which parts of the cached data are missing with the new VM-host mapping. Then we copy or move them to the new hosts. With the second option, the data need not be moved, but we may create load unbalance in the system due to the affinity of VMs for a set of physical servers. If the "preferred" physical machines of the "return" client do not have enough resources left, the system still has to put VMs into other available physical machines. In our implementation, we selected the second approach. However, one can employ the delaying scheduling approach [138] to achieve both locality and fairness.

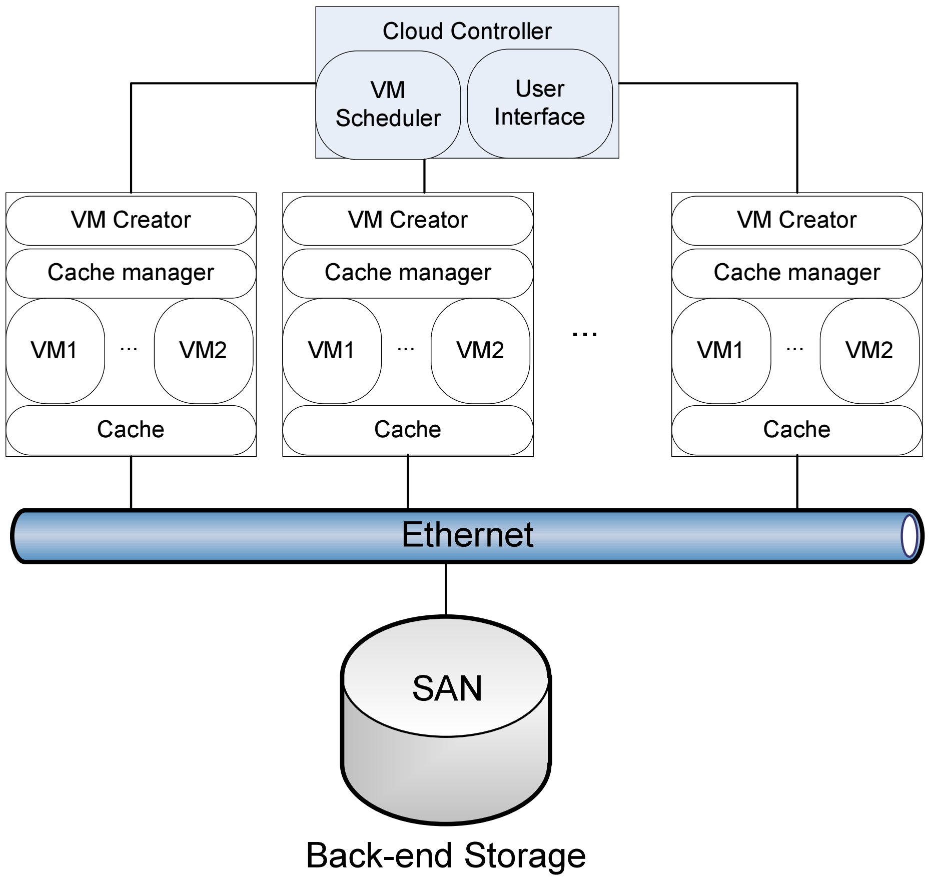


Figure 6.8: The logical view of the system.

Figure 6.8 describes the logical view of the system. It is easy to recognize that we employ the traditional master-slave model (it also matches with Eucalyptus, on which we will implement our prototype). The master node (called front-end in Eucalyptus) is the Cloud Controller in the figure. It contains the web service interface and the VM scheduler. The scheduler communicates with the nodes to start VMs on them. Each node also uses web services to handle requests from the scheduler. On the nodes, we have running VMs, the cache partition (system cache), the VM creator (xen, kvm, libvirt) and the cache manager. The cache manager attaches the cache to the VM and also implements the replacement algorithms such as FIFO and LRU.

When a user submits a job, the VM scheduler system would try to allocate her VMs to the physical hosts that already contained her data. Otherwise, if she is just a new user who hasn't uploaded any data to the system (we mean HDFS, not S3),the system can schedule her VMs to any available hosts that trigger none or the smallest portion of the cache replacement process.

To ensure the isolation requirement, the cache is allocated at separate partitions on the physical hosts. It is attached automatically and accordingly (for appropriate users) to the VMs during the VM creation process and persists after VM termination. Another design-related decision to make at this time is if each user has a separate partition in the hosts or if all of them share the same partition as a cache. Again, due to security and isolation concerns the design of one user for one cache partition was chosen.

The size of the cache partition is an important factor. It can be varied or fixed. For simplicity, we chose the fixed size cache for each user (user cache), but the partition for the cache at a physical machine (system cache) can be dynamic. However, it is worth noting that with the same physical machine, it can host different number of VMs depending on the VM types (m1.small, c1.large, etc). The user cache size is also proportional to the VM types too although it is fixed. Different VM types have different fixed user cache sizes. To ensure fairness, all users have the same size of cache for the same type of VM.

Size of the system cache is also important. If it is too small, it does not reduce the data movement efficiently because the system has to replace old data with the new one. The old data does not have many chances to be reused. If it is too large, the storage portion for the ephemeral storage of the VMs (regular storage of the VMs) is reduced. Hence so is the number of VMs available to be hosted on a physical machine. In fact, the cache size depends on the system usage parameters such as average number of users and the user cache size. If the system has too many users at the same time or the user cache is too large, it can trigger the data replacement too many times. We can extend the size of the system cache or reduce the user cache size to relieve this.

There are two cache replacement levels in our system. The first one happens at the physical node, and the second one is at the whole system. When the local cache is full, the node should migrate the data existing in its cache system to other available machines. When the global cache is full, it is time to actually remove the old cache following the traditional policies such as FIFO or LRU. There are two options available here: remove all data of a user (users) or remove just enough data to accommodate the newest user data.

The operation of the cache replacement mechanism depends on two factors: cache availability and VM availability. These two factors are independent from each other. A physical host may not be able to host new VMs, but it still can have available cache and verse versa.

The flow chart of the scheduler and cache replacement mechanism at the whole system level is expressed in Figure 6.9. The detailed algorithm of the cache replacement at the local level is not shown here to save some spaces. It implements FIFO and LRU policies and is triggered when a new user comes in; the node has available resources to host his/her VM but does not have any user cache slot left.

In the Elastic MapReduce model of Amazon, the users only need to specify the jobflow which includes the execution file, input and output data from S3 and the parameters together with the total number of VMs (including Hadoop master and slaves). The Elastic MapReduce system automatically allocates the requested number of machines, starts Hadoop, stages in the

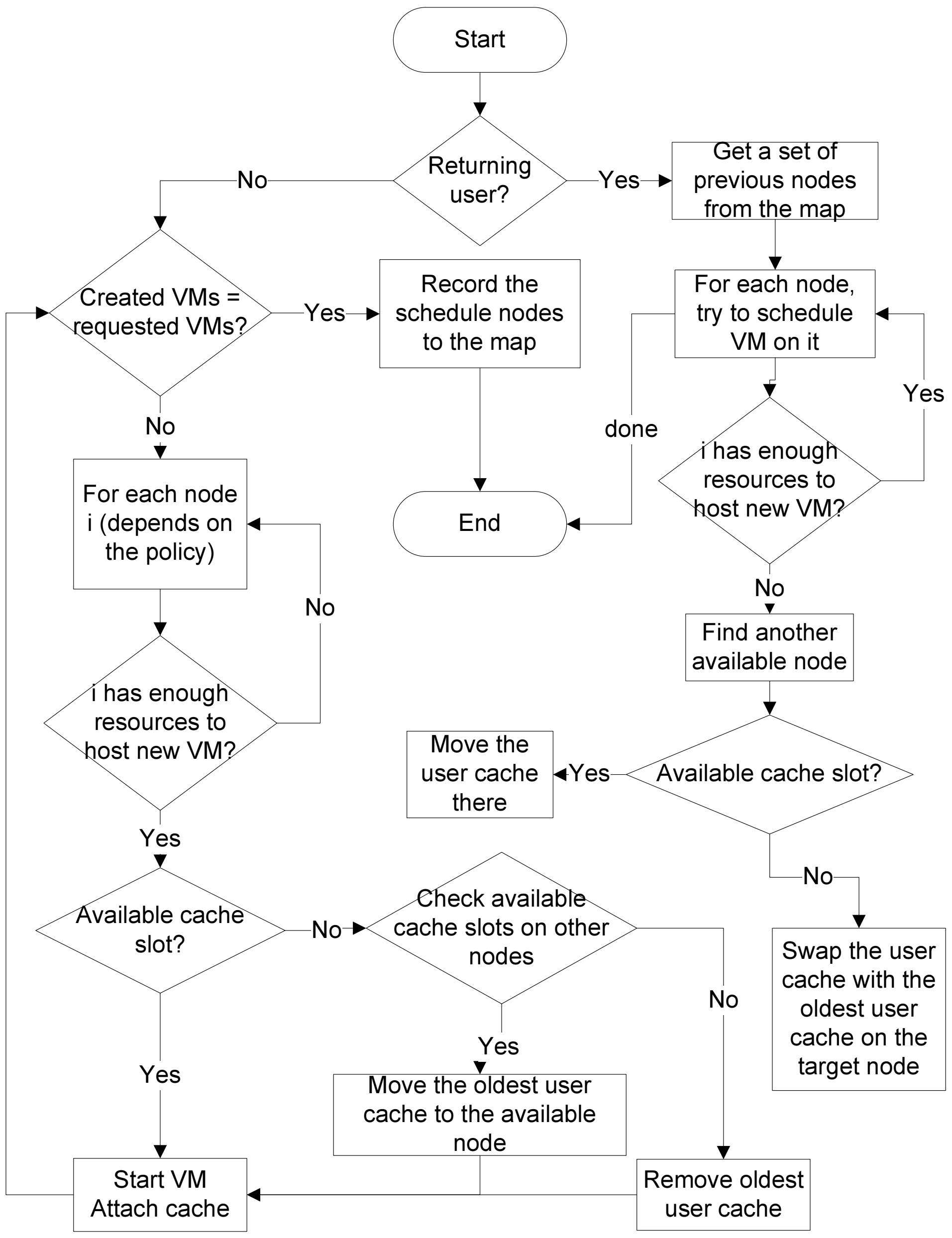

Start
Returning user?
No
Yes
Get a set of previous nodes from the map
Created VMs = requested VMs?
Yes
Record the schedule nodes to the map
For each node, try to schedule VM on it
Yes
done
i has enough resources to host new VM?
No
For each node i (depends on the policy)
End
No
Find another available node
i has enough resources to host new VM?
Move the user cache there
Yes
Available cache slot?
Yes
No
Available cache slot?
No
Check available cache slots on other nodes
Swap the user cache with the oldest user cache on the target node
No
Yes
Yes
Move the oldest user cache to the available node
Start VM Attach cache
Remove oldest user cache

Figure 6.9: The flowchart of the scheduler and the cache replacement at the front-end.

data and execution files, executes them, stores the results back to S3 and tears down the VMs. Software such as Whirr [53] automates the process of deploying the Hadoop cluster and will be discussed later.

With our cache and scheduler, EMR users do not even need to specify the number of VMs they need again (they just need to specify it the first time) since the system would schedule them with the previously specified number of machines. However, if they do intentionally want to change this, our system simply tries its best to schedule as many VMs close to their data as possible and let Hadoop take care of the process of redistributing data automatically since Hadoop was designed to tolerate such high churn. Actually, Hadoop has already taken care of the joining and departing of new VMs (nodes). If the user want to add more machines (VMs), the load balancer distributes data to them. If the user reduces the number of machines, Hadoop will enter the safemode if there are not enough replicas as configured.

6.4 Implementation

6.4.1 EMR vs. Private Cloud

Since we do not have the insight of EMR, we use a wildly-used open source private Cloud to measure the steps involved in the jobflow. Figure 6.10 shows the execution relationship between our Cloud and EC2 by comparing the execution time when running the same application (CloudBurst) on the same set of data in EC2 and Eucalyptus with the same VM configurations. We used the same number of VMs with almost the same configurations and ran the same application on the same set of data. The correlation between our system and EMR is 0.85.

In Figure 6.10, our system has worse performance than EMR because of limitation in our network. According to [3, 130] the bandwidth between EC2 instances (computing nodes) is 1Gbps (i.e. 100 MB/s) and between EC2 and S3 is 400Mbps (i.e. 50 MB/s). In our network configuration, the average bandwidth between the physical nodes (computing nodes) is only 2.77Mbps, between computing nodes and the persistent storage node (Walrus) is about 2.67Mbps and between client and the front-end is 1.2Mbps. In addition, the CloudBurst ap-

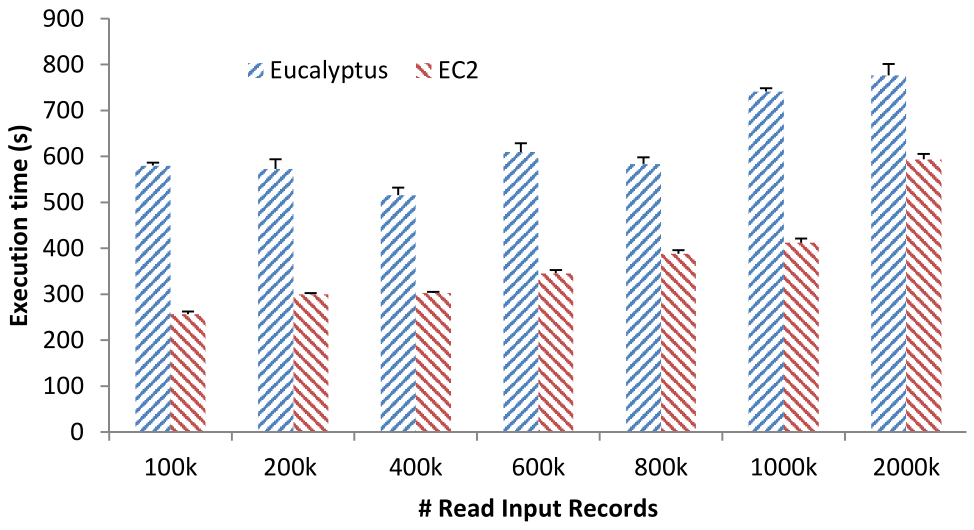


Figure 6.10: EC2 VS. Eucalyptus.

plication itself is a network-intensive application [103]. Therefore, the Amazon EMR outperformed us with this particular experiment. However, we only want to show with this experiment that our system has a strong correlation with EMR so that we can use it from now on.

6.4.2 Our Implementation

In our implementation, we use Hadoop version 0.20.2 and Eucalyptus version 2.0.3, and the EMI image is CentOS.

To add our cache system into Eucalyptus, we need to create a cache partition on each physical node, identify the fixed size for each type of instance, add it to the VM, and mount it to the suitable point in the VMs so that Hadoop can access it. In our prototype, the fixed size of the user cache is only 1.2 GB. For simplicity, we also only ran experiments with one type of instance.

As mentioned in the previous section, although the user cache is fixed, the system cache is not. To enable the dynamic size in system cache partition, we use the logical volume manager (LVM). To add the user cache to a VM, we modified the VM xml creator script of Eucalyptus to attach a cache as a block device in VMs. It is worth noting that doing this does not actually mount the block device (the cache) to the file system; therefore, it has not been able to be used yet.

To mimic EMR, we need to create a script receiving information like jobflow. This script then invokes other scripts to create instances, start Hadoop, run the job and terminate the instances. Since the standard Eucalyptus VM image (EMI) does not have Hadoop, we modified it to install and configure Hadoop, and enable it to run the user script too. This allows us to automate the starting/stopping script easily and to mount the user cache to the directory used by HDFS.

Due to the restriction on our network, we chose the static option for Eucalyptus network configuration. The network topology is shown in Figure **??**. The Cloud can only be accessed from the front-end, and we use a separate machine to act as the user who submits the jobflow.

So far we have only mimicked the EMR and prepared the cache storage at the node. The main part of our system is to schedule users' VMs close to their data. When a user comes back to the system, the data should already be in the Hadoop cluster. To realize this, we modified the scheduler of Eucalyptus. However, before discussing more detail about the modification, we will look at the current VM scheduler of Eucalyptus.

In Eucalyptus, the scheduler is located at the cluster controller on the front-end server. Currently, there are three scheduling policies available: Greedy, RoundRobin, and PowerSave. Greedy chooses the first physical node that has enough resources to run the VM. RoundRobin loops over the nodes one after another until it finds a node that can host the VM. PowerSave schedules VMs on the first awake machine it finds. If the remaining resources on the awakened machines are not enough to host the new instance, the system powers up a sleeping machine and starts the new instance there. When a node does not host any VM, it enters the sleep mode. The default policy is RoundRobin.

We added code to the cluster controller of Eucalyptus to record the map between users and their VM locations. Then we modified the VM scheduler so that if it sees the same user coming back, with the same number of instances requested (by looking at the recorded map), it schedules the user's VMs to the previous locations (if possible) following flowchart 6.9. If such locations have no resource left for the new VMs, the system moves the corresponding cache to other available nodes (using default schedule policy) and starts the VMs there. If there is no available cache slot in the whole system, we should swap the cache in the original node with the (oldest or randomly) one on the "available to run VM" node. However, we have not implemented the cache movement among nodes. Also, when removing the cache we do not maintain the meta data to know what part of the data of which user was deleted so that later, when that user comes back, the system can automatically fetch the removed data back. It is noteworthy here that this situation is different from the case in which the user intentionally reduces the number of instances. In this case, the cache system deletes the data, not the user. Nevertheless, the user is aware of this.

In our prototype, we implemented the local level cache replacement at each node controller following the algorithm shown in the previous section. To implement the LRU and FIFO, we utilized the uthash library [64].

6.5 Performance Evaluation

6.5.1 Experiment Setup

We have implemented a prototype cache system on Eucalyptus. Our testbed consists of 12 machines. One of them is the front-end server (containing Cloud Controller, Walrus and Cluster Controller). Another one is a machine outside the cloud acting as the client machine. The last 10 machines are the computing nodes (Node Controller). The detailed configurations of these machines are shown in Table 7.2; and the overview of the system network is shown in Figure **??**. In the figure, the server on the leftmost is the client machine. It communicates with the front-end of our private Cloud via the Internet connection. Our front-end server has two network cards. One is for internal communication with computing nodes and another is for external communication. The workload used in this section includes CloudAligner, CloudBurst, the Hadoop Sort benchmarks and Grep. All of them were mentioned earlier except

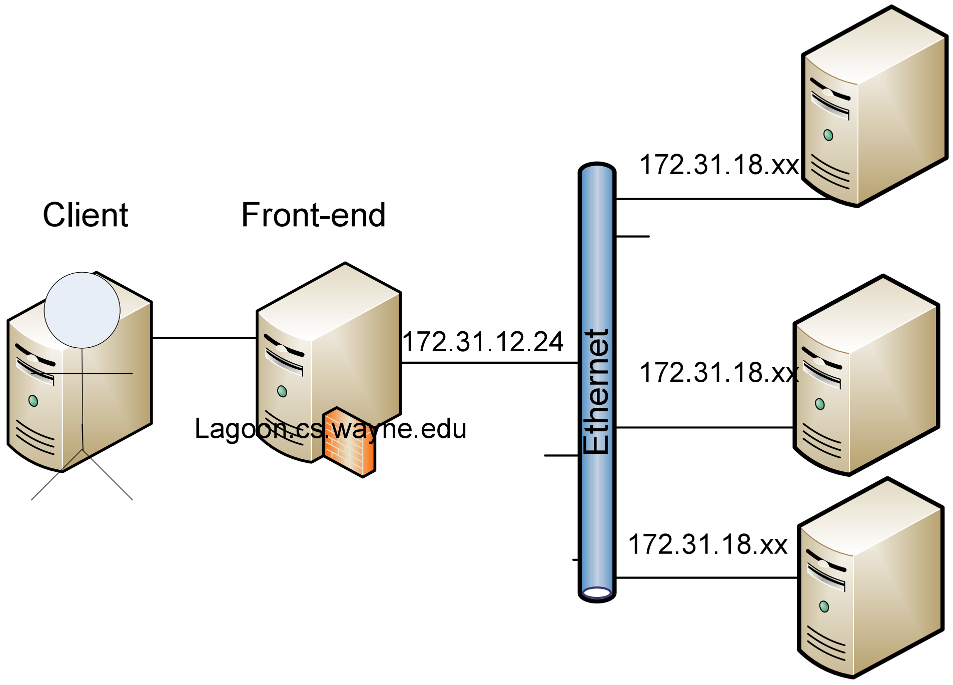


Figure 6.11: The network configuration of our private cloud.

Type	**Num.**	**CPU**	**Memory**	**HDD**	**OS**
Client	1	AMD 2GHz	6GB	250GB	Ubuntu
Front-end Server	1	AMD Phenom II	8GB	500GB	CentOS
Computing node	10	Xeon 2.80GHz	2GB	40GB	CentOS

Table 6.1: The testbed configuration.

Grep. Grep is also a built-in MapReduce application of Hadoop. It is used to search for an expression on the input files. The input data of Sort is generated by the RandomWriter. The input for Grep is composed from the log files of Hadoop and the Shakespeare play. The search expression is ’the*’.

	100k	**200k**	**400k**	**600k**	**800k**	**1000k**	**2000k**
Writable	4.3M	8.6M	18M	26M	35M	43M	86M
Text	5.2M	11M	21M	32M	42M	53M	106M

Table 6.2: The size of the input read files.

The data for both CloudBurst and CloudAligner is composed of two pieces. One is the read sequences produced by the sequencer such as Illumina GAII, HiSeq 2000 or Pacific Biosciences, etc. For our experiments we obtained the real data from the 1000 Genomes project. Particularly, we used the accession SRR035459 (956MB in size). From that file, we extracted some subsets to use in our experiments. The size in bytes and in number of input splits (the first row) are shown in Table 6.2. In the table, row 2 expresses the size of the data in the MapReduce Writable format, and row 3 expresses the size of the original text file. Another piece of data is the reference genome. For this data, we chose chromosome 22 of the human genome (with the original text size of 50M and 9.2M in the MapReduce Writable type). It should be noted that these data are small in size. They are used only for the proof of concept here. The real data are much larger. For example, there are 22 chromosomes in the human genome, and, normally, the alignment software needs to align the read to all of them. Moreover, the size of chromosome 22 is the smallest among human chromosomes. For example, the size of chromosome 2 is 237MB.

6.5.2 Soundness

The first step is to verify the correctness of our system. To do this, we create a new user, create new instances with that user, record the locations of the VMs, move the data into the HDFS and execute a jobflow. Then we turn off all the VMs and run the jobflow again. The

VMs should be scheduled on the same physical nodes as recorded. To verify that the data is already in the system, we used the “ls” command and executed the same job with the same input parameter without moving the data from the Walrus.

6.5.3 Improvements

To show the improvement in the performance when using our cache system and the VM scheduler, ideally we should run experiments on data stored in Walrus and compare the result against the one in HDFS because with the warm cache the data already exists in the HDFS. However, the jets3t library (version 0.6) employed in the version of Hadoop that we used doesn’t support communication with Walrus. The newer version of jets3t does support Walrus, but we have failed to integrate it to work with Hadoop. As a result, we applied the following two methods.

First, the data is moved from the Walrus server to the HDFS, and the data movement durations are recorded. Without the cache, the data always has to be moved in; therefore, the total execution time of a jobflow should have this data movement time added. This method is called *addition*. Figure 6.12 shows the performance of CloudBurst and CloudAliner when using and not using cache. As shown in the figures, when the size of the data increases, especially after 800K, the gap between using and not using cache is wider because the increasing rate of data movement time is faster than that of processing time. Figure 6.13 (for the Sort application) and 6.14 (for the Grep application) support this observation too. Basically, if the data is large and the processing time on the data is small (due to the MapReduce framework), the benefit of using our cache is more significant. For example, in the Sort experiment, moving 1GB of data to the system takes 6795s with our network but sorting it takes only 1405s.

Second, we compare the result when running the same application with data in HDFS and data in Amazon S3. The computing nodes in this case are in our testbed, not at Amazon. In other words, we use our local cluster to process the data from HDFS and from S3. This method is called *HDFS_S3*. The execution times of CloudBurst and CloudAligner with dif-

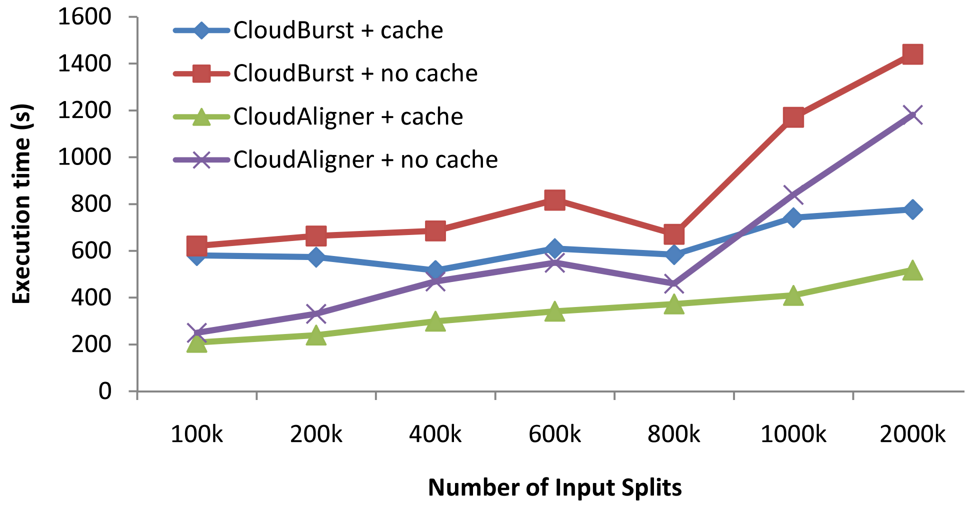


Figure 6.12: Performance of the CloudAligner and CloudBurst applications with and without warm cache in the addition method.

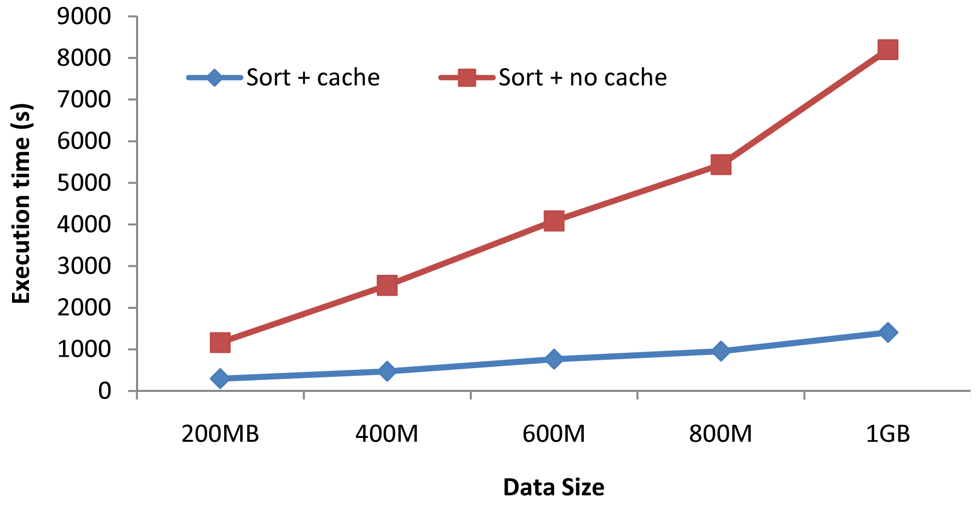


Figure 6.13: Performance of the Sort application with and without warm cache in the addition method.

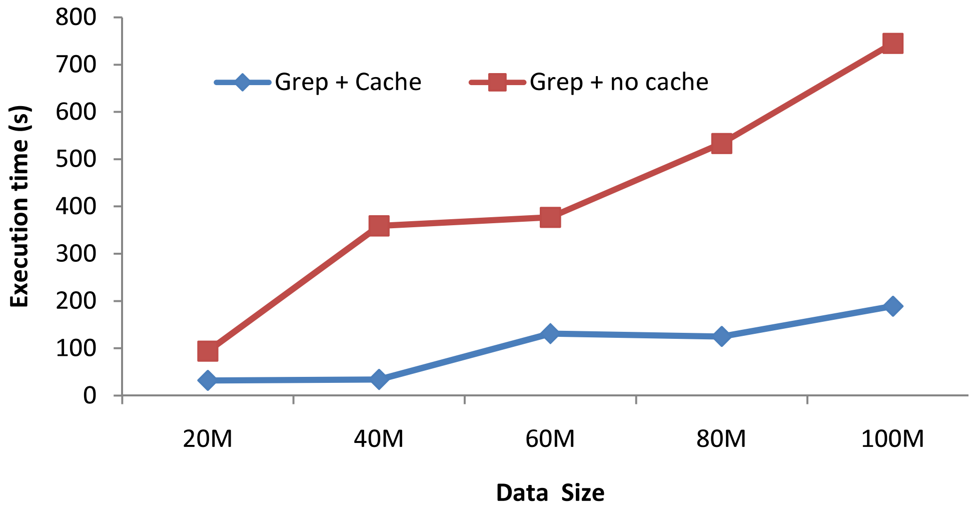


Figure 6.14: Performance of the Grep application with and without warm cache in the addition method.

ferent configurations are expressed in Figure 6.15. While the three figures in the first method reveal the actual values of the time to move data and the time to process the data, Figure 6.16 and 6.17 show only the relative performance of applications without cache and with warm cache. This helps us to see clearer how many percentages do we improve with our system. With CloudAligner and CloudBurst, our system can improve its performance up to 56.4% and 47.4%. The performance gain in Sort and Grep is more significant because they only need less time to process larger data. In particular, the Sort application can benefit up to 75.1%, and the number of the Grep application is 83.7%. This result makes sense because all Grep does is just to cut the large data into small parts and to search each part in parallel for the pattern linearly.

Besides the execution time, we also plan to measure other traditional statistics such as the cache hit and miss as well as the effects of FIFO and LRU replacement policies. Such metrics can be obtained by using simulation on usage traces of production systems. Unfortunately, not only don't we have access to them, the traces do not have enough information. For example, the information needed to compute cache miss/hit is which blocks of data that each job of a user accesses, but the traces from Yahoo and FaceBook [38] contain no such information.

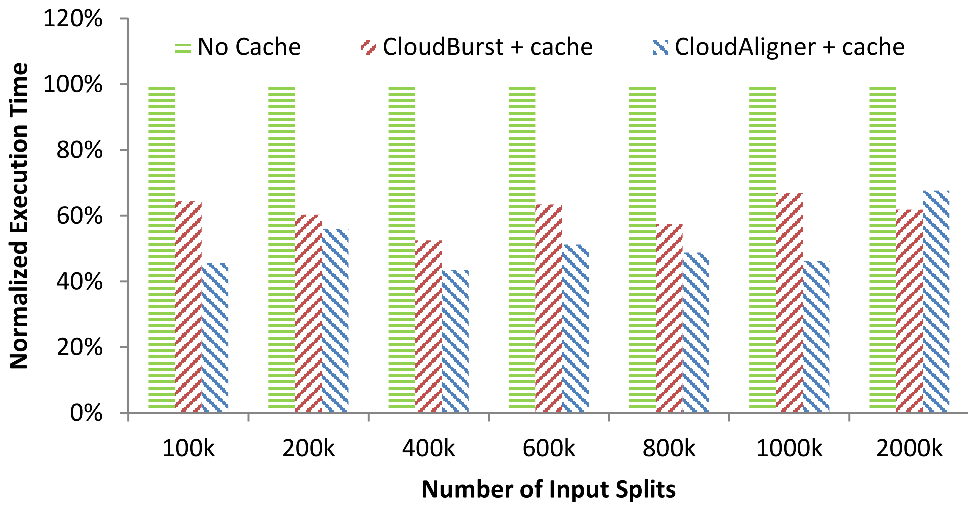


Figure 6.15: Performance improvement of the CloudAligner and CloudBurst with warm cache in the HDFS_S3 method.

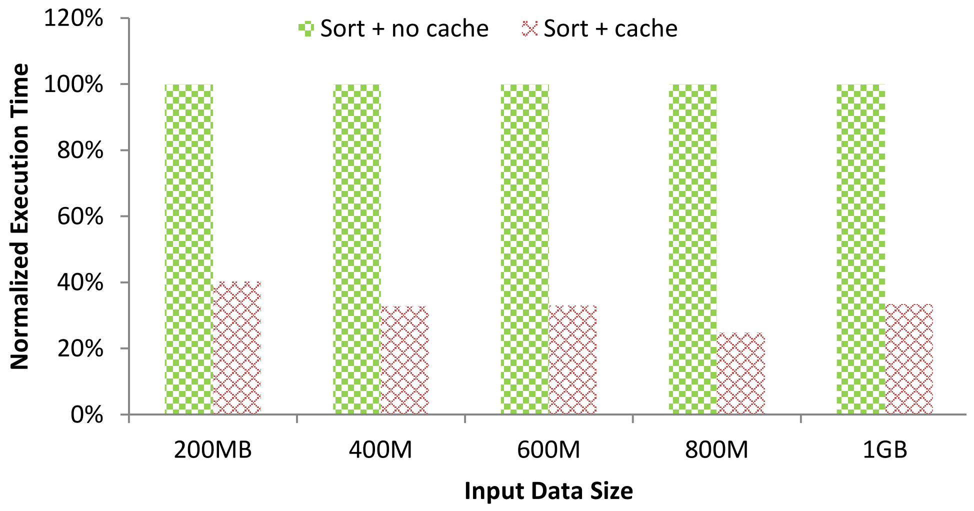


Figure 6.16: Performance improvement of the Sort application with warm cache in the HDFS_S3 method.

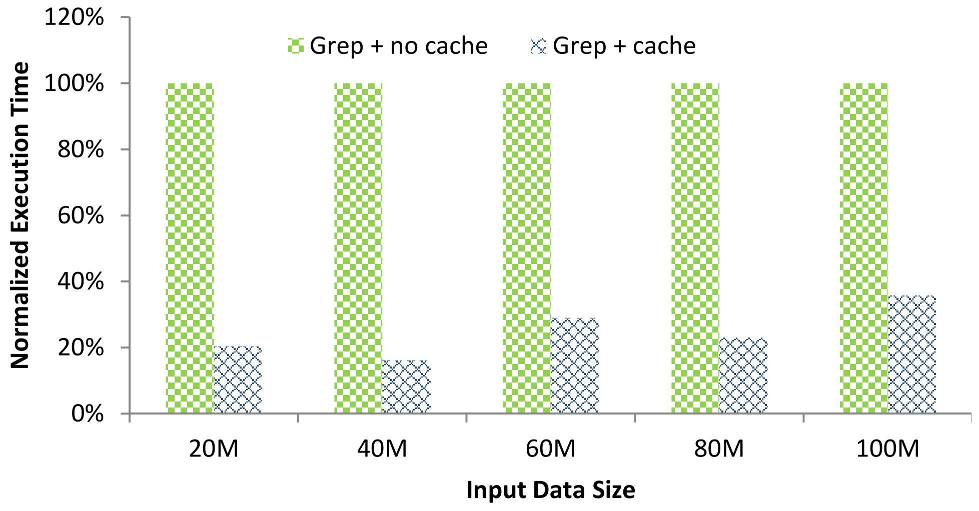


Figure 6.17: Performance improvement of the Grep application with warm cache in the HDFS_S3 method.

6.5.4 Overhead Analysis

In comparison to the old system, we introduced new images, added a cache partition to the VM, mounted it, started hadoop and modified the scheduler. There are three main places that we modified: the cluster controller, the node controller and the VM. In terms of the startup time of the VM, the overhead may arise from the difference between the modified and the original EMI, which is the installation of Hadoop, the VM startup xml (libvirt.xml) and the mounting script. This is negligible because the installation only needs to be done once, and the size of the image does not change after this installation; the mounting script contains only one Linux mount command; and the VM startup xml has only one additional device. In addition, the startup time of the VM is very small in comparison to the time to prepare for the instance (copy root, kernel, ramdisk, etc files from the cache or from Walrus, create ephemeral file, etc).

In terms of execution time, the overhead is also negligible because as shown in the flowchart in Figure 6.9, our additional scheduler is active only when the user has already visited the system. In addition, our scheduler even improves the performance of the scheduling process since it does not need to spend time on iterating through each node to find the available one for the user request.

6.6 Related Works

There is a distributed cache system built-in Hadoop [140]. However, our cache is different. With the Hadoop distributed cache the files are already in the HDFS, while in our case, the files are in the back-end storage system (S3). The data in [140] is cached between the map tasks and reduce tasks. In addition, while the target platform of [140] is the small cluster, ours is the Cloud.

Amazon Elastic Block Store (EBS) is a good fit to use for our cache as they persist independently from the instances. However, the current EMR does not support it. To use EBS, the users have to manually create suitable AMI with Hadoop, start it, attach the EBS, configure, start Hadoop, run job, etc. Whenever they want to run their job again, the whole process has to be repeated manually though the data persists on the EBS volumes.

Dealing with the performance of MapReduce has received much attention recently. For example, argued that Hadoop was designed for a single user, [121] and [137] proposed solutions to improve the performance of MapReduce in multi-user environment.

6.7 Summary and Future Work in Internal Data Movement

In conclusion, in this chapter, we have shown several existing issues with the current MapReduce system in the Cloud environment. Then, we have proposed a distributed cache system and a VM scheduler to solve the data movement issue. The side effects of doing so is that we can improve the performance of the system. We have implemented the prototype of the system with Eucalyptus and Hadoop. The experimental results indeed show a significant performance gain with certain applications.

For the next step, we plan to make Hadoop work with Walrus, obtain the real trace to measure the cache hit or miss ratios, implement the cache movement across the whole system and monitor the removed data.

CHAPTER 7

EXTERNAL DATA MOVEMENT

In genomics, the volume of data to be processed is normally very large. Recently, the next generation sequencing (NGS) has generated data at even a higher rate with lower cost which makes the volume of data increase exponentially. Unfortunately, many of the existing software tools in the field were developed without taking into account the size of data. They assume that the data is fit in a single powerful machine, limiting their applicability and algorithms from two perspectives: resource capacity and performance (cannot run in parallel). As a result, they cannot handle very large data. Recognizing this limitation, recent work has been employed cloud computing and MapReduce to store and process big data. There are three main steps involved in handling data in the cloud: (1) moving data from users to the cloud, (2) processing it and (3) sending results back to users. The actual time that the user has to wait before she receives the results should be the sum of the data transmission time t_{trans} and the processing time t_{proc}. As the size of the data reaches a certain scale, the t_{trans} becomes the dominant element. However, most studies have focused only on improving the processing time t_{proc} and assumed that the data is already stored in the cloud, which is not true as many organizations now produce their gene sequence data on their own and store it locally. In this chapter, we propose an innovative mechanism called SPBD: Streamlining Processing for Big Data (SPBD is pronounced as ”speed”) to improve the total waiting time of end users called respond time. The evaluation on the prototype of SPBD shows that it can reduce the respond time of the end user up to 34% in the traditional wordcount application and up to 31% in the metagenomic application.

Figure 7.1 describes the relationship of the work (SPBD and Metagenomic) in this chapter with others. It is worth to note in the figure that in this chapter, we also introduce another MapReduce conversion of an existing popular application in genomics research area: metagenomic.

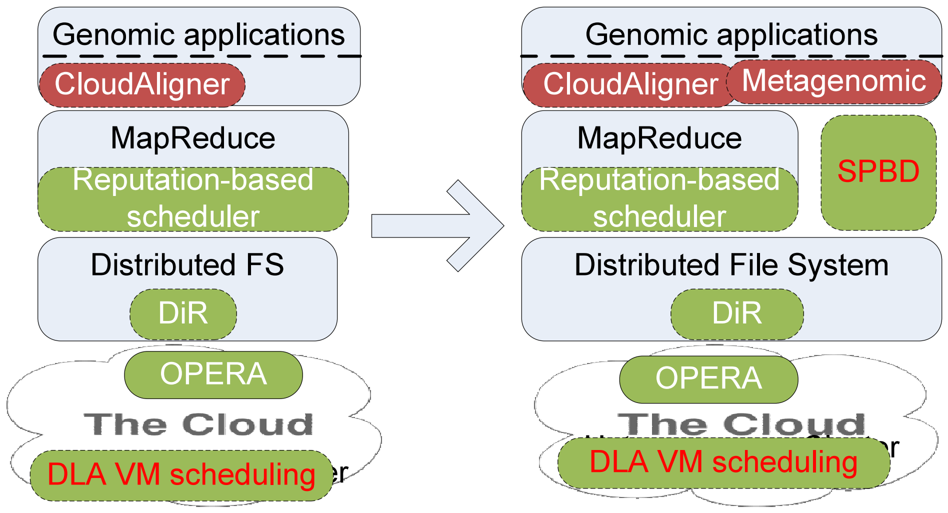


Figure 7.1: External Big Data Movement.

7.1 Introduction

Thanks to the recent significant improvement of technology in genomics, researchers now can sequence significantly more data with lower price. As a result, the NGS data is often very large causing challenges in storing and processing it. For example, at our C.S. Mott center, the Illumina HiSeq sequencing system [11] can generate about 250 millions of reads which is about 60 GB with a single run. The sequencer normally has 8 lanes, each can handle at most 12 samples. In total, we can have 96 samples or 96 read files, 60GB each (pair-end read) for one run. It takes about several days to generate the data and two days with the state of the art alignment software (novoalign [12]) to process these data.

Without very strictly concern on the security, using Cloud solution to store and process such large data set appears to be a good solution since it provides virtually unlimited highly available data storage, unlimited and elastic computation power without up-front cost.

In terms of computing model, usually, when dealing with large data set, parallelization is the most effective and widely used solution. Recently, Google MapReduce [49] and Microsoft Dryad [73] are among the most wildly used frameworks to parallelize genomic applications [117, 82, 108, 103]. Although adopting Cloud and parallelization is not easy for the biology scientists who has little expertise on computer science, recent work such as [24, 80] has somehow lowered down the barrier. In addition, bioinformaticians who are equipped with skills in both computer science and biology are supposed to find no difficulty in using Cloud and parallel programming models or frameworks.

The problem is that, in practice, the data is often generated in a place that is different from where it is processed and analyzed [21]. Therefore, the procedure to obtain the analytic result of the data in reality is usually as the following:

- The data is moved to the data center
- The analysis software is executed against the data inside the data center or the cloud or the local cluster
- The result is downloaded to the local machine of the scientists

In order to transfer data to the remote sites, the popular methods such as the "scp" shell command, ftp client, web (http) are often used by many bioinformaticians. Even when the data is getting larger, such methods are still applied. Most bioinformaticians seem not aware of other methods such as GridFTP. One of the big and annoying issues with transferring large file over the network is the failure. Traditional data transfer methods do not handle failure gracefully.

Up to this point, we have described the current general situation in handling large data in genomics. In this work, we put our focus specifically on the MapReduce model in the Cloud such as the Amazon Elastic MapReduce (EMR). We also select the Hadoop implementation of the MapReduce as our working framework. The applications are therefore of the MapReduce compatible class only.

After narrowing down our context, we are going to explore how data is moved to Hadoop, in particular, to HDFS. Traditionally, the most popular method with the hadoop users is to use the “hadoop dfs -put ¡filename¿” or distcp shell command of Hadoop to move or copy the data from a local machine to the HDFS. With the Amazon EMR, data is moved to the Amazon S3 first using many tools ranging from command line interface (cli) such as s3cmd and S3-Util to the graphical user interface (GUI) such as CloudBerry Explorer, Bucket Explorer, etc. Then, a “job flow” is typically created using either the GUI-based AWS Management Console or the cli-based Amazon Elastic MapReduce Ruby Client to execute a MapReduce application with the uploaded data.

With large data, Amazon suggests to use their physical import/export service. The users can store up to 4TB of data in an eSATA or USB 2.0 portable hard drive and have it couriered to the nearest Amazon site. Besides, in general, there are many other tools such as Pentaho [14] and Apache Sqoop to transfer data to/from Hadoop (HDFS).

We have discussed so far the state-of-the-art in processing large data in genomic and chosen the target system. It is time to identify clearly the problem and the objective of our work. Most of current work in the targeted context focus on either improving the processing time of the software [117, 82, 103] or the data transmission time only [102]. In this work, we aim at minimizing the respond time (t_{res}) which is measured from the time the users start uploading their data to the cloud to the time the results are ready to be sent back or downloaded. We do not take into account the transfer time of the result from the computing facility (cloud) to the users because in our genomic context, such the size of the results is often much smaller than that of the input data. For example, after the alignment and format adaptation stages, 70GB of input data can generate only approximately 5GB output. For example, after the alignment and format adaptation stages, 70GB of input data can generate only approximately 5GB output.

Many people may argue that we should only focus on the processing time since the data need only to be transferred at most once. Once the data is at the data center, the user will run many applications/software on it many times. Therefore, if we can reduce the execution time of such tools, it is more beneficial. Nevertheless, based on our observation in reality, in researches related to the large NGS data, the alignment or mapping tools, which directly deal with the large data in the pipeline/workflow mentioned in [114], is normally run only once or twice due to the lengthy execution and high cost (if running in the commercial clouds). Consequently, minimizing the respond time (t_{res}) is very crucial and useful as well.

As we can not avoid moving data from users to the data center, we propose to process the big data in a streamlining way. Our approach is to split the data from users into a bunch of small parts and streamline them to the data center. Right after the first part arrives, the system starts processing it. The system keeps receiving the new subsequent parts while processing the arrived parts. After all the parts are processed, the system merges all the results and then sends it back to the users.

To expound our idea, we implemented a prototype of SPBD and evaluated it comprehensively with two MapReduce-based applications: wordcount and metagenomic. However, since the metagenomic software was originally developed to run locally, it cannot handle data larger than 1GB on a 8GB of RAM machine. We converted it to MapReduce style to make it scalable. With this new version, the software virtually does not have the limitation on the size of the input data. The results shows that SPBD can improve the respond time of wordcount and metagenomic 34% and 31% with 32GB of input data, respectively.

In summary, in this work, we will try to minimize the respond time when executing the MapReduce-style data analytic software on the large NGS data set in the Cloud with Hadoop framework. The overview of our target system is shown in Figure 7.2.

The main contributions of our work include:

- Convert the traditional metagenomic application into MapReduce style to achieve high scalability
- Design and implement of SPBD that can significantly reduce the respond time and therefore cost when processing large data in cloud
- Comprehensive evaluation and analysis of SPBD

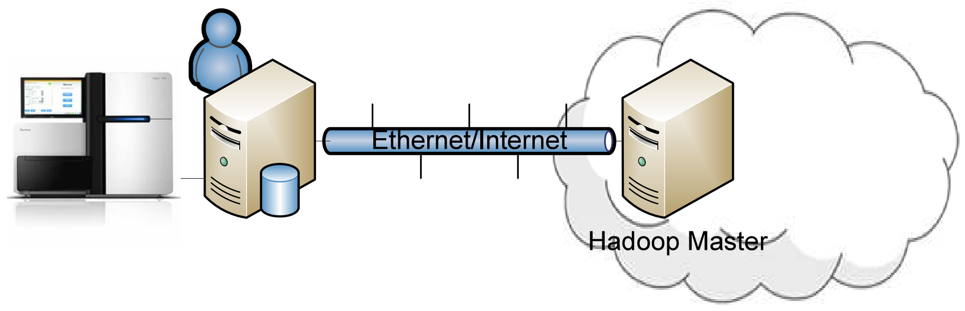


Figure 7.2: System Overview

The rest of the chapter is organized as the follows. The next section will describe our approach in detail with problem formation and examples. After that, Section 7.3 will provide the design and implementation of SPBD. Following that is the evaluation section which describe the experimental results and finding when employing SPBD. The last section concludes our work and discusses about the remaining as well as future work.

7.2 Our Approach

The main idea of our approach is to adapt the pipeline processing model in the CPU to our scenario. A CPU instruction in the pipeline model is equivalent to a complete processing of a partition of data in our context denoted as P_i where i is the number of the partition. From now on, we will use partition and chunk of data interchangeably. Each P_i only has two phases (transmission and processing) or stages rather than many stages like the CPU instruction. For example, the Itanium 2 CPU has 8 stages. Besides, all stages of a CPU instruction have the same clock cycle (time to complete a stage) while they have arbitrary time in our case.

In our case, instead of waiting until after the whole data is moved to the Cloud to start the analytic application, we start the analytic process as soon as the first chunk of data arrive. While the data of the first chunk is analyzed the second chunk is received. When the processing of the first chunk of data is complete, the second chunk of data should be ready to be processed. While processing the second chunk of data, the system receives the next one. This process continues until the last chunk of data arrive. Depending on a specific application, an additional

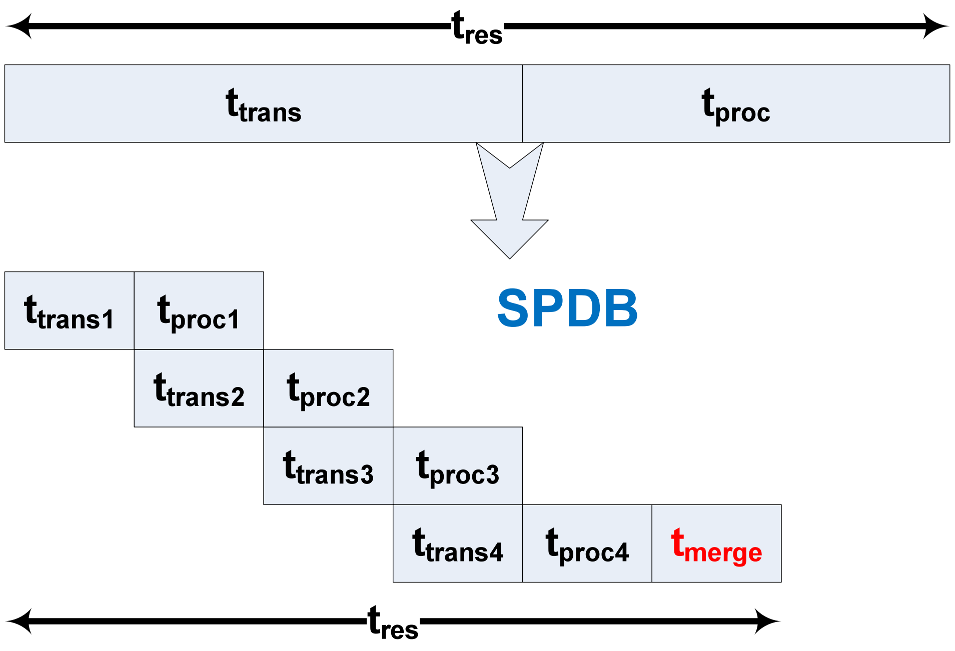


Figure 7.3: Streamlining Proceesing

“merging” process can be run after the last chunk of data is processed or as soon as the first chunk of data get processed. The merging process is needed to merge the output results of the execution of all chunks. Although defined by application developer, such merging process is usually rather simple in practice. We will show that in our case study. Figure 7.3 summarized the whole idea clearly. In the figure, t_{trans} and t_{proc} are respectively the data transmission time and processing time. Also, t_{res} and t_{merg} are the respond time and merging time (the time to execute the merging process) respectively.

The question here is that how large is each chunk of data to achieve the optimal respond time? or how many partitions should we cut the input data into? To answer it, we need to find out how to compute the respond time. Although the stages have the same size in the Figure 7.3, it is not necessarily true in practice. In the overlapping periods (where the receiving and processing processes take place at the same time), any stage required more time to execute would be used to compute the respond time. Let us consider two possible cases: $t_{trans} > t_{proc}$ and $t_{trans} \leq t_{proc}$. Let N be the number of partitions of the input data. For simplicity, we

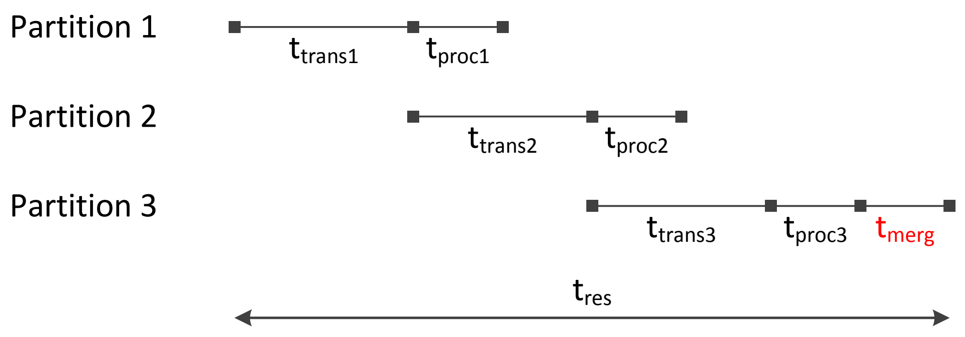


Figure 7.4: Respond time calculation when $t_{trans} > t_{proc}$

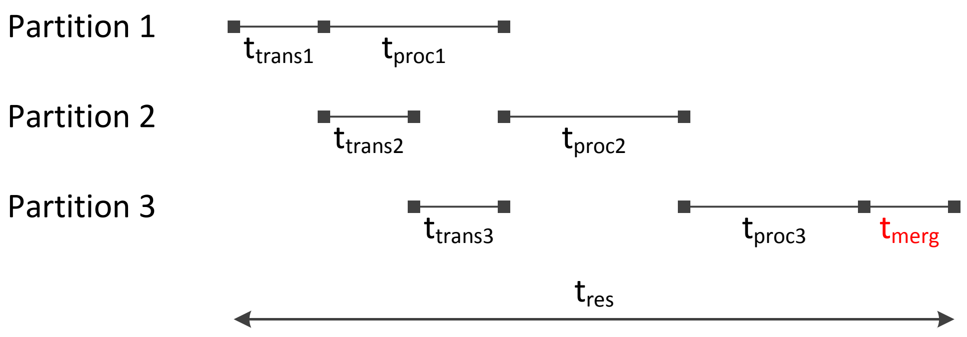


Figure 7.5: Respond time calculation when $t_{trans} \leq t_{proc}$

assume that all partitions have the same size and the t_{trans} as well as t_{proc} of a partition are the same with those of others. The merging period contributes to the calculation of the t_{res} only after the last chunk of data is processed. All of its executions (if any) taken place before that are in fact "masked" by either the data transmission or the processing stage.

If $t_{trans} > t_{proc}$. As illustrated in Figure 7.4, t_{res} can be computed as

$$\begin{aligned} t_{res} &= t_{trans} + (N-1) \times t_{trans} + t_{proc} + t_{merg} \\ &= t_{proc} + N \times t_{trans} + t_{merg} \end{aligned} \tag{7.1}$$

Since the transmission time dominates in this case, the system always finish processing all the received data and waits for the next one.

If $t_{trans} \leq t_{proc}$. Similarly, from Figure 7.5, t_{res} can be computed as

$$t_{res} = t_{trans} + N \times t_{proc} + t_{merg} \tag{7.2}$$

In this case, from the figure, we can see that even the data is ready to be processed but the system has to wait until the process of the previous chunks finishes since the processing time is larger.

In both cases, the transmission of the next chunk of data is started as soon as the previous one finishes. In general, the respond time is calculated as

$$t_{res} = min(t_{proc}, t_{trans}) + N \times max(t_{proc}, t_{trans}) + t_{merg} \tag{7.3}$$

For example, giving that the total transmission time of the whole data $t_{trans} = 150$ and the processing time of the data $t_{proc} = 100$, the respond time with the traditional method (process after receiving the data completely) is $t_{res1} = t_{trans} + t_{proc} = 250$. Assume the input data is divided into 2 partitions with $t_{trans} = 80$, $t_{proc} = 60$ and $t_{merg} = 10$ for each partition, from equation 7.3, the respond time in this case is $t_{res2} = t_{proc} + N \times t_{trans} + t_{merg} = 60 + 2 \times 80 + 10 = 230$. If the data in cut into 3 partitions where t_{trans}, t_{proc} and t_{merg} are 55, 40, and 12, respectively, the respond time now is $t_{res3} = 40 + 3 \times 55 + 12 = 217$. Table 7.1 shows the summary of all of these examples.

Table 7.1: The Respond Time Calculation Examples.

N	t_{trans}	t_{proc}	t_{merg}	t_{res}
1	150	100	0	250
2	80	60	10	230
3	55	40	12	217

It is worth to note that all t_{proc}, t_{trans} and t_{merg} in equation 7.3 are the function of N. As the number of partitions N increases, both t_{proc} and t_{trans} decrease, but t_{merg} increases. In summary, t_{res} is also a function of N. Our problem now becomes a simple optimization problem: *"Given a fixed input data, finding the number of partitions N so that t_{res} is minimum"*.

However, in reality, the problem is how to identify all the components in the equation giving that all the system gets from its users are the data with known size and an analytic application. This will be addressed in Section 7.3.

7.3 System Design and Implementation

7.4 System Design

In this section, we are going to discuss about our system called SPBD that realized our idea. The inputs for our system are an input data, a data analytic application and a path to a local folder to store the output. The data is located outside the cloud, at user's site. The applications are already in the cloud. Like Amazon EMR, we can assume that there are a pool of analytic applications in the cloud for user to choose. This assumption makes sense because in genomics, researchers often have a collection of applications to select from to analyze their data such as Galaxy, BLAST. Moreover, even if the user would like to use a special or in-house application, it should not be a problem since the time to upload the new application is negligible, and it can be reused many times.

Originally, we assume the data analytic software is a "partitionable" MapReduce application which is a MapReduce job that can process input data in parallel (i.e there is no dependence among different chunks of input data). However, this is in fact the property of all MapReduce style jobs since all map tasks are independent. As a result, it is possible to state that our method can be applied to any MapReduce job.

The output from our system is also the output of the selected applications. The system takes the input data from user, transfers it to the cloud, executes the selected applications there and sends the results back. From the users' point of view, they only need to provide the path to the local input and output folder and the names (together with other parameters) of the applications. The progress of their jobs/applications in the cloud can be monitored via the built-in Hadoop web site. Once the jobs finish, the output data is typically written to HDFS or Amazon S3. However, our system can send it back to user if they want.

To identify all the components in the equation 7.3, there are two approaches: static and dynamic. In the static approach, the functions of the terms of the equation 7.3 are fixed which means, for example, if the $t_{trans} = \frac{1000}{N} + 20$, it remains the same all the time. In the dynamic approach, such assumption is released. This is more realistic since the status of the network is not identical at different time point due to network congestion or the variety of the network traffic. Also, in the cloud environment with the virtualization, the execution time of the same applications with the same size of data may not be similar at different time points due to the differences in the status of the cloud infrastructure (virtual machines, load, network) and differences in the content of the data itself.

Let us revisit the equation 7.3 since all t_{proc}, t_{trans} and t_{merg} are not directly a function of N. They are in fact the functions of the size of the partition. Therefore, we need to change their variable from the size of the partition to the number of partitions or verse versa. Let p is the size of a partition (assume that all partitions have the same size), S is the size of the whole input data. The relationship between p and N is

$$p = \frac{S}{N} \tag{7.4}$$

Replacing this equation to all terms of equations 7.3, we now indeed have t_{res} as a function of N or p.

Since t_merge is not trivial to estimate in practice, we remove it from equations 7.3 in our implementation. In our scenario, the input data is partitioned and processed by a specific analytic application. After that, the output of all partitions goes to the merging process to produce the final result. Therefore, it is difficult to capture the relationship between t_{merge} and the input data size since it in fact depends heavily on the output of the analytic applications. Later, in the evaluation section, we will show that the form of t_{merge} of different applications are different.

Since we assume that the processing time and transmission time is linear with respect to the size of the input data, with out lost of generality, we denote

$$\begin{cases} t_{trans} & = & ap + b \\ t_{proc} & = & cp + d \end{cases} \tag{7.5}$$

In our context, a and c are always greater than 0 since t_{proc} and t_{trans} are monotonic increasing functions of input data size. The partition size p is also greater than 0. To obtain the minimum of t_{res}, we need to know when t_{proc} is greater, smaller or equal to t_{trans} to remove the max and min functions in equations 7.3.

We have two following cases:

- **Case 1:** $a > c$

  The equations 7.3 becomes

$$t_{res} = \begin{cases} ap + \frac{dS}{p} + b + Sc & p < \frac{d-b}{a-c} \\ (\frac{S}{p} + 1)(ap + b) = (\frac{S}{p} + 1)(cp + d) & p = \frac{d-b}{a-c} \\ cp + \frac{bS}{p} + d + Sa & p > \frac{d-b}{a-c} \end{cases} \tag{7.6}$$

  – When $p = \frac{d-b}{a-c}$ it is trivial to find the minimum of t_{res} because the t_{res} is a constant.

  – When $p < \frac{d-b}{a-c}$, t_{res} is minimum when

    ∗ if $d \geq 0$

      $p = \sqrt{\frac{dS}{a}}$

    ∗ if d¡0 we do not have minimum in this case. However, this is not likely to happen since we did not consider t_{merge} which is of the form

      $t_{merge} = eN + f = e\frac{S}{p} + f$

      with e¿0 and is rather large value since when the number of partitions increases, the merging time increase too.

  - When $p > \frac{d-b}{a-c}$, t_{res} is minimum when $p = \sqrt{\frac{bS}{c}}$ when $b > 0$

- **Case 2:** $a < c$

  The equations 7.3 becomes

$$t_{res} = \begin{cases} cp + \frac{bS}{p} + d + Sa & p < \frac{d-b}{a-c} \\ (\frac{S}{p}+1)(ap+b) = (\frac{S}{p}+1)(cp+d) & p = \frac{d-b}{a-c} \\ ap + \frac{dS}{p} + b + Sc & p > \frac{d-b}{a-c} \end{cases} \tag{7.7}$$

  Similar to the previous case, we can derive the value p to minimize t_{res}.

After having p at each subcases, we calculate the value of t_{res} accordingly and compare them to get the global minimum value of function t_{res}.

At this stage, we can further fine-tune p by monitoring the network status and the analysis progress. For example, the network keeps changing all over the time, so the transmission time of the coming partition is most likely different from the previous ones.

7.5 Implementation

SPBD is implemented in Java with Client-Server model. The main idea is that the client keeps sending data and the server decides when its buffer has enough data (size of the partition) to start processing to achieve optimal overall respond time. The problem is how many partitions/chunks should the input data be divided. In other words, when should the system starts processing a chunk of data?

The client simply provides an interface for user to specify their job (application names and parameters). The path to the local input data is included in the parameters. The job specification is similar to that of the Amazon EMR ruby client. Other information such as the ip address of the server, credentials, access keys, etc is specified in a configuration file. Before sending the large input file, the client sends the job specification and the data size to the server. After receiving the acknowledge message from the server, it starts transferring the input data using the TCP protocol.

The server is implemented with two approaches: static and dynamic. In the static approach, we execute all available applications in the pool with different sets of data and different sizes off-line to identify all the functions (using linear regression method) of the components in equation 7.3. In the next section, we will show that the linearity assumption is hold in practice since the R-squared values are very close to 1 which indicates a good fit model.

In the dynamic approach, the components of equation 7.3 are identified online, i.e. when executing the job. At the server, the input data is drafty divided into two parts with nearly the same size. The first part is used to "learn" the t_{proc} and t_{trans} functions. The second one is processed in a streamlining way using the computed optimized size. The administrator (at the server side) specifies a number of samples when starting SPBD service. It is used to identify different sample sizes in our linear regression process. The init_size (the size of the first sample partition) can be computed from the input size as $init\_size = \frac{S}{2^{SAMPLE\_NUM-1}\_2}$. The size of the i^{th} sample partition is a double of the size of the $(i-1)^{th}$ sample partition. The algorithm in the receiving function at the server is shown in Algorithm 7.5. With this approach, we can see that it is difficult to "learn" the function of t_{merge} since the sizes of partitions are different. After deriving such functions, we stored them for future fine-tuning and analysis.

In addition, we further fine-tuned SPBD by keeping monitoring the t_{proc} and t_{trans} after computing the optimized size. If, in the case that the optimal value is achieved when $t_{proc} = t_{trans}$, $|t_{proc} - t_{trans}| > threshold$, we update the optimized size accordingly. For example, if $a > c$ and t_{proc} is greater we decrease the size of the partition.

In the current implementation, SPBD does not send the result data back to users since it is rather simple. If using the Amazon EMR, the results are stored back to S3 and there are many graphical easy-to-use tools to interact with S3 as if it is a local file system.

Algorithm 2 SPBD Algorithm

```
Require: In: the size of the input data S, init_size and SAMPLE_NUM
  $partition\_size = init\_size$
  $accummulated\_size = 0$
  repeat
      bytesRead = Read from socket to buffer
      write buffer to HDFS
      accummulated_size+=bytesRead
      partition_num=0
      while $accummulated\_size \geq partition\_size$ do
          record $t_{trans}$ for the received partition
          create and submit a new job asynchronously to Hadoop to process the received partition
          if partition_num¡SAMPLE_NUM-1 then
              partition_size*=2;
          else if partition_num==SAMPLE_NUM-1 then
              Start a new thread to compute the optimized size for a partition
              the optimized size will be used to update the partition_size
          end if
          accummulated_size=0
          partition_num++
      end while
  until $BUFFER\_SIZE! = bytesRead$
```

7.6 Evaluations

We evaluate SPBD with two applications: WordCount and Metagenomic. WordCount is selected because it has served as a typical example in the MapReduce tutorial and therefore is good the demonstrate the insights of SPBD. We hope it can help users to adopt our system easier. Another purpose of choosing it is to show that our approach is not necessarily limited to the Genomic applications only. Metagenomic is chosen among many genomic applications since it deals with large data sets. In addition, its outputs are fix-sized matrices which can demonstrate the fact that the merging process is very different among applications. We will discuss about this later in this section.

Our testbed configurations is described in Table7.2. The local network speed is 100Mbps. The computing cluster is running Hadoop version 0.20.203.

Type	**Num.**	**CPU**	**Memory**	**HDD**	**OS**
Client	1	3.6GHz	24GB	72TB	Solaris
Master	1	2.4GHz	16GB	850GB	CentOS
Slaves	4	2.4GHz	16GB	¿320GB	CentOS

Table 7.2: The testbed configuration.

Table 7.3 gives the information about the lines of code of the merging part that the developer needs to write to use SPBD. We can see that it is rather simple to write such part. Most likely it is just a slightly modification of the reduce part in the original applications. The following two

Table 7.3: The Lines of Code of the Merging Part.

WordCount	MetaGenomic
76	152

subsections describe our experiment results in detail. We intentionally did more experiments with the wordcount to get the insight of SPBD since it is more popular than the metagenomic.

7.6.1 WordCount

Wordcount is a MapReduce application incorporated in the example jar file of the Hadoop distribution. It is used to count the number of occurrences of all words in an input file. The input data used in our experiments is the data dump from wikimedia [16].

Experiments, which produce data in Figure 7.6 and 7.7, are used to show the relationships between the size of partitions and the processing as well as transmission time of a partition with the same input data. The two figures also include the linear regression function together with the R-square values of the processing time and transmission time. The size of the input data in Figure 7.6 and 7.7 is 1GB and 2GB, respectively. The figures show that with the same input data (the same content), if it is partitioned into different sizes, both t_{proc} and t_{trans} of each partition can be modeled with the linear regression. It is noteworthy that at each data point in the figures, the input data in cut into partitions with the same size.

The figures also show that the linear functions obtained with an input data may be different from those with other input data.

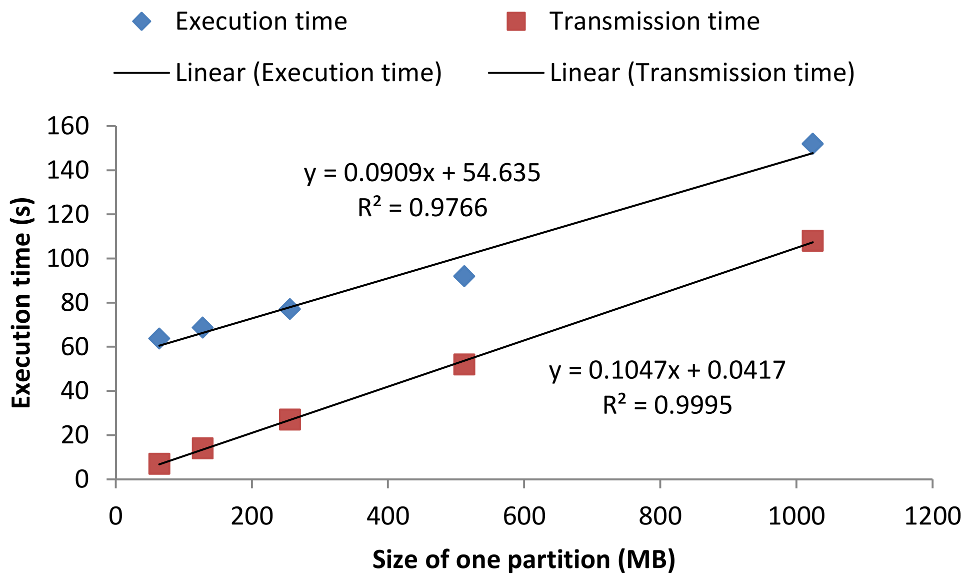


Figure 7.6: The t_{proc}, t_{trans} and their linear regression functions with data input of 1GB.

Figure 7.8 displays the execution time of the merging process with 1GB and 2GB input data. It illustrates that with the same size of input, if the number of partition increase, the merging time would increase too.

Figure 7.9 is used to show the relationship between input data size (not the partitions) with

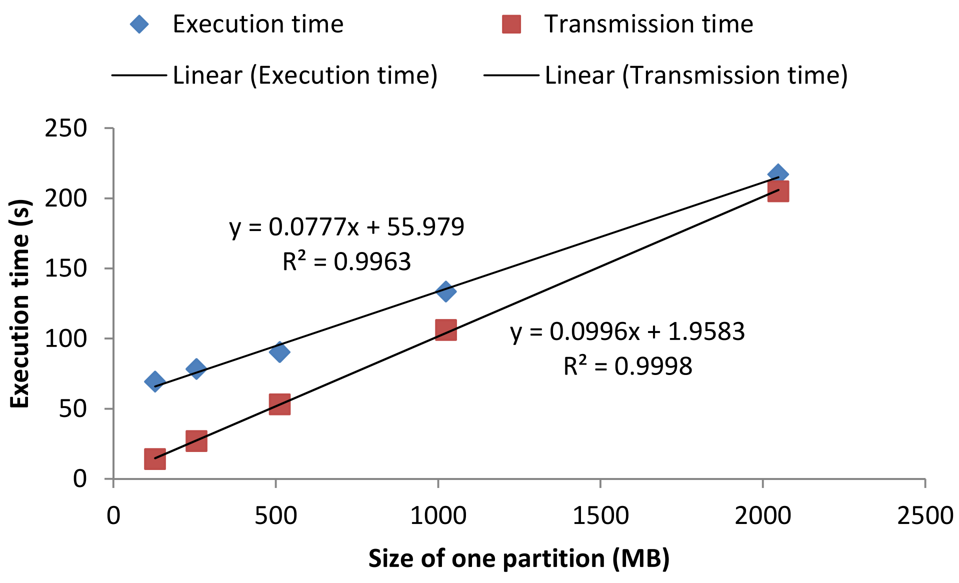


Figure 7.7: The t_{proc}, t_{trans} and their linear regression functions with data input of 2GB.

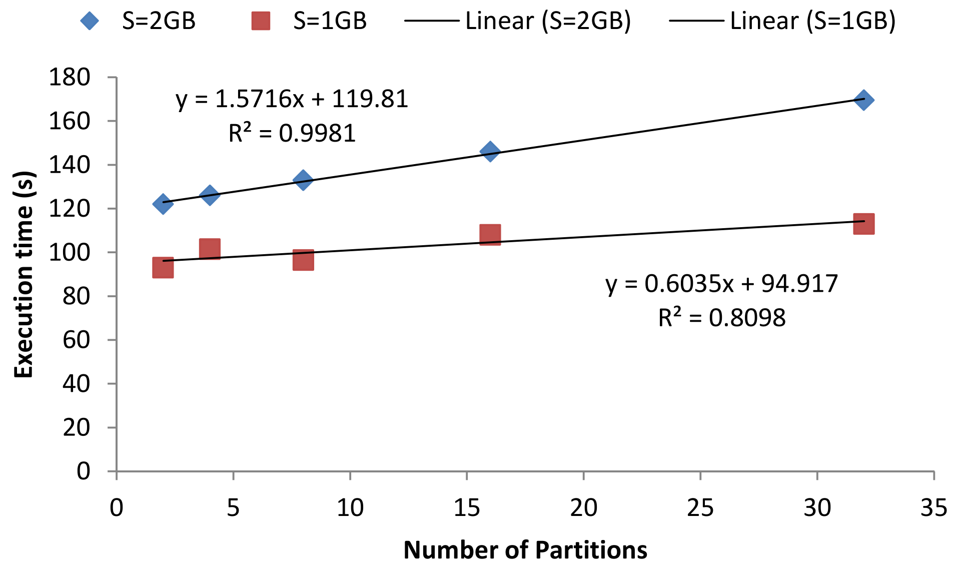


Figure 7.8: The linear regression functions of the merging process with data input of 1GB and 2GB.

the t_{proc} and t_{trans}. The content of all input data in this figure are different.

It is worth to note that the data size in Figure 7.9 is in GB while in Figure 7.6 and 7.7 are in MB. Therefore, the linear function parameters are significantly different. Also, the content of input data used in Figure 7.6, 7.7 and 7.9 are different between each other. The values obtained in those experiments are the average of several executions.

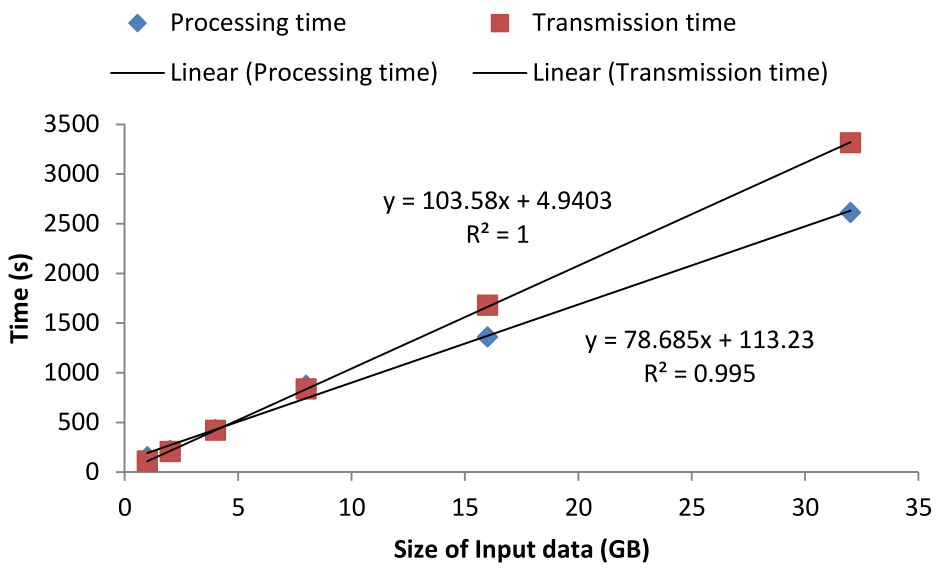


Figure 7.9: The linear regression functions of WordCount with different sizes of data.

The main purpose of the above experiments to verify that the relationship between the size of input data as well as the size of partition and the t_{proc} as well as t_{trans} is linear regardless of the content of the data. Another conclusion is that the functions of t_{proc} and t_{trans} derive from the linear regression are different among executions. Consequently, we should not use the same computed optimized partition size between different runs.

After verifying the linear assumption, we conducted experiments with our SPBD system. We mimicked the real scenario by the following setup. The input data is stored in the client machine (see Table 7.2) which act as the storage server at the client side. This machine also contains SPBD client code. The master node runs the SPBD server code and accepts the job submissions. It is also the master node of Hadoop. The the rest slave node are the computing node. In reality, both master and slavers are in the cloud and the connection from them to the client is through the Internet. However, in our experiments, all of them are in the same local network. This setting in fact does not affect the applicability of SPBD since SPBD learns the functions through measurement.

In Figure 7.10, we evaluate SPBD with the static approach on different sizes of a partition. In this experiment, we ran wordcount on 32GB of data. We can see that, with the same input data, different partition sizes can result in considerably different respond time.

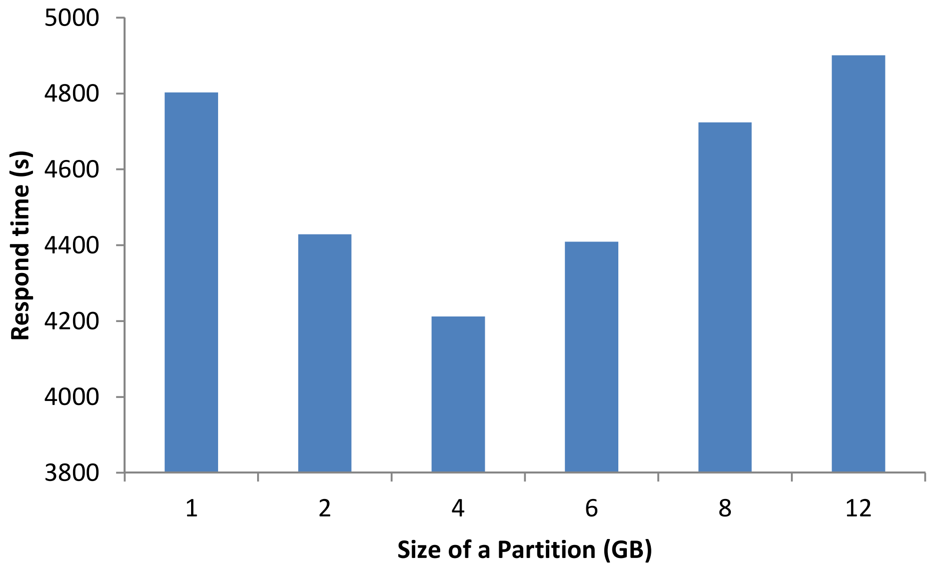


Figure 7.10: The respond time of WordCount with the static approach and different sizes of a partition.

The last experiment is to study the benefit of using SPBD. In this experiment, we use the dynamic approach with SAMPLE_NUM=3. The “NO-SPBD” configuration is computed by setting the partition size equal to the size of the whole input data. It is also noteworthy that the SPBD values already include the merging time. In other words, the results of the output in the two configurations are the same. The results are expressed in Figure 7.11. It can be seen from the figure that as the size of input data increases, SPBD provides more improvement. For example, with the input of 32GB, the respond time with SPBD is only 66% of the existing approach.

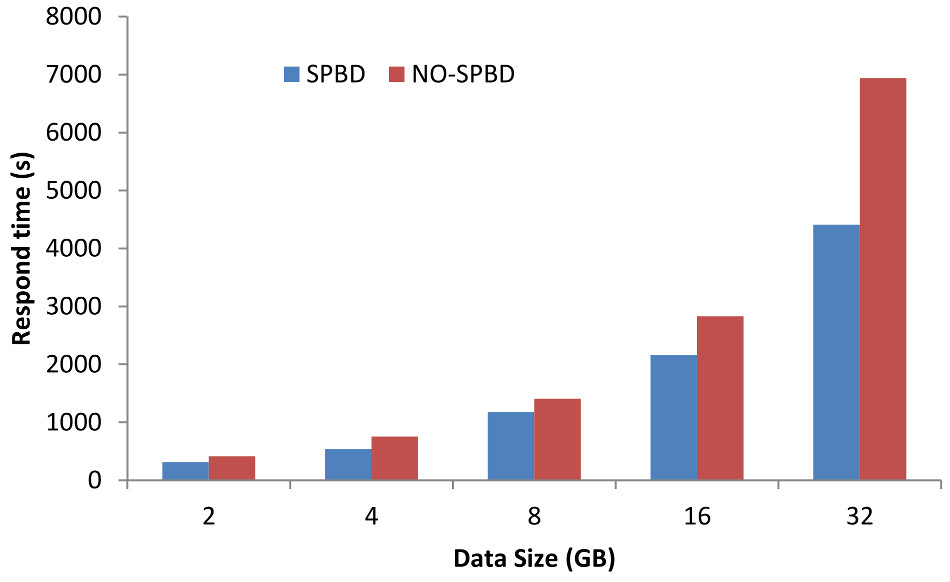


Figure 7.11: The linear regression functions of WordCount with different sizes of data.

7.6.2 MetaGenomics

Metagenomics is the study of genetic material sampled directly from the habitats of microorganisms [133]. Microbial DNA Sequencing helps us to study unknown microorganisms that cannot be cultured and thus cannot be sequenced. In analyzing metagenomics data, one main objective is to recover the taxonomic composition of the sample. Next generation sequencing (NGS) has made metagenomics easier by providing more data, but better clustering methods is required due to shorter read length and higher throughput.

In this experiment, we employed a metagenomics application developed by a bioinformatic research group in our department. The application is implemented with the popular K-Mean clustering technique. As mentioned earlier, the original application is not in MapReduce style and is also not scalable. Therefore, we rewrote in into MapReduce style to help it run with larger data set.

The input data for it is generated using MetaSim [112]. This availability of input data is also one main reason to select this tool for our experiment.

Similar to the previous experiments on wordcount, in this sub section, we also verify the linearity in the relationship between the input data and t_{proc} as well as t_{trans}, study the property of the merging process and finally estimate the benefit of using SPBD.

Figure 7.12 and 7.13 respectively describe the values of t_{proc} and t_{trans} measured with different input data sizes. They also include the results by applying linear regression on these data. Similar to the wordcount experiments, the linear assumption is also hold in this case.

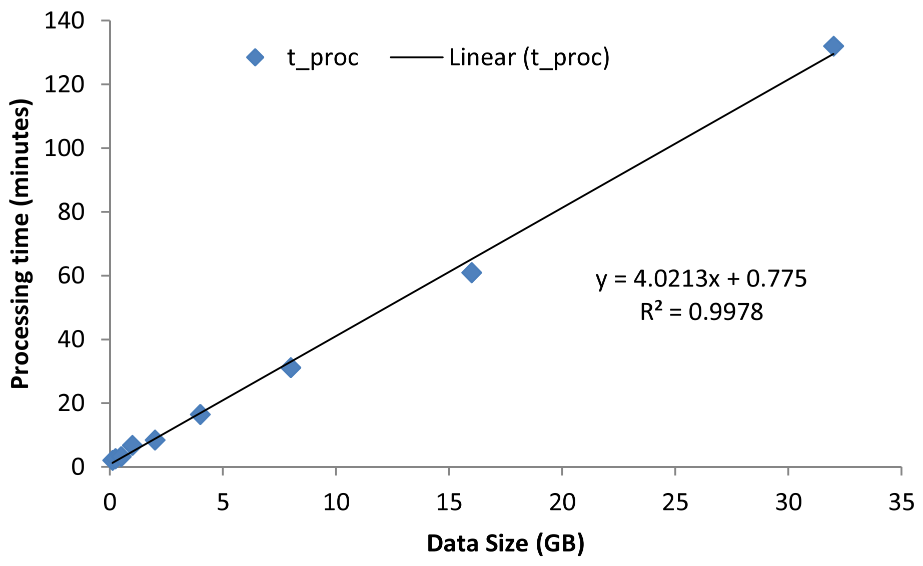


Figure 7.12: The measured processing time and its linear function.

The execution time of the merging process on different size of data is depicted in Figure 7.14. This time, the merging time is almost constant which is different from that in the wordcount experiments. This is because the output of the metagenomics application is just a small number of small-size matrices independent to the size of the input data. Therefore, combining them in parallel taking only small and almost constant amount of time. This experiment also demonstrates that the merging time is difficult to estimate and it heavily depends on particular applications.

Lastly, we ran the metagenomics application with SPBD and without SPBD on different data set. At the input of 32GB, the respond time of SPBD is improved by 31% in comparison to the “NO-SPBD” approach.

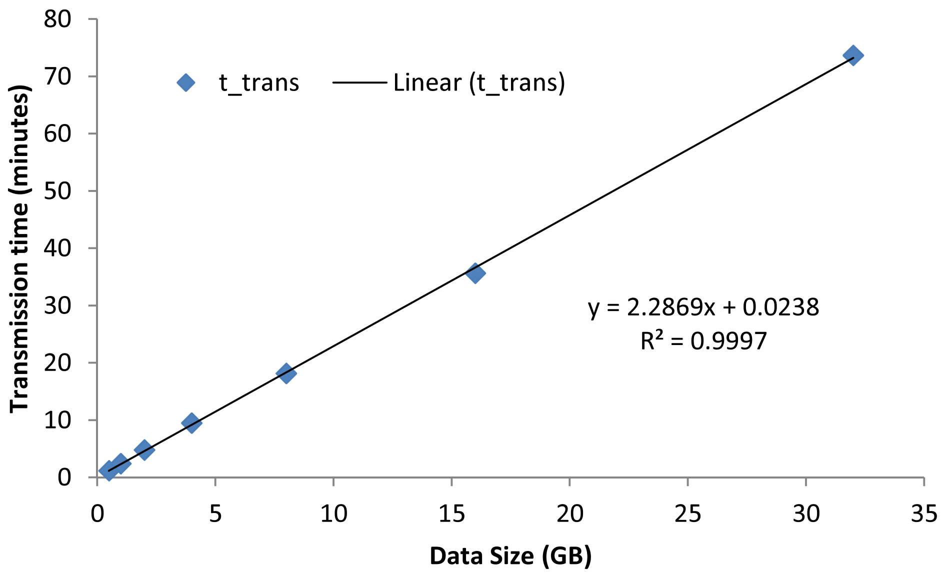


Figure 7.13: The measured transmission time and its linear function.

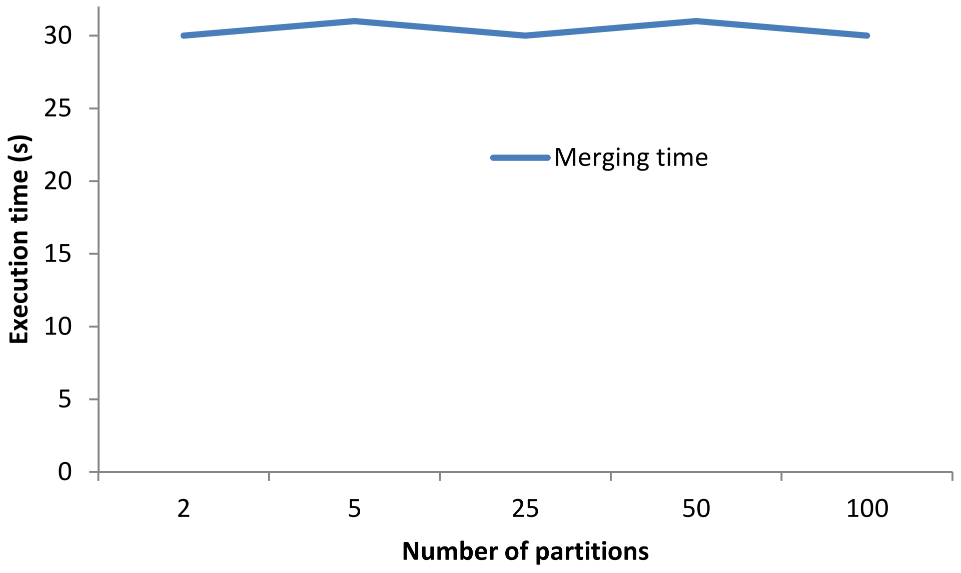


Figure 7.14: The measured merging time.

It is interesting to note that with input data smaller than 8GB, the performance of SPBD is a little bit worse than the “NO-SPBD” configuration. This indicate that SPBD is only beneficial with big data. With a certain application, if the input size is smaller than a certain value, it is good to upload them all to the cloud and processing it afterward. Currently, we set this value equal to 1GB.

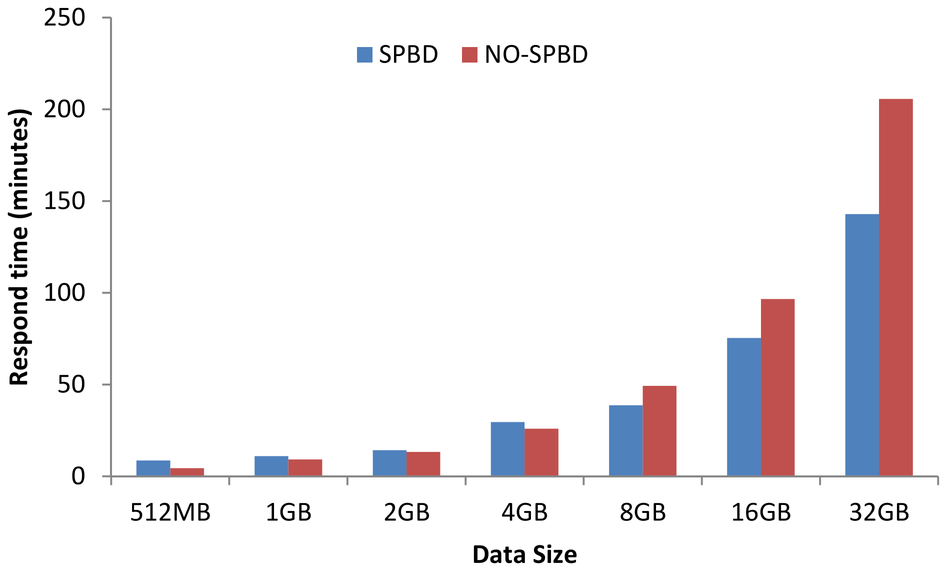


Figure 7.15: The benefits of using SPBD.

7.7 Summary and Future Work in External Data Movement

In conclusion, in this chapter, we have discussed the state-of-the-art in analyzing large data sets in genomics which is combining cloud solution with a data parallel framework such as MapReduce or Dryad. Then we have proposed, implemented and evaluated SPBD which is a system that automatically transfers and processes large data in the MapReduce cloud. The experiment results have proved that SPBD can improve the respond time notably.

Our method considers the context that with a certain set of data, the user only runs their analysis tools in the cloud once since our idea is to incorporate the processing time with the data transmission time. If the user re-executes the analysis on the same data, our approach is inapplicable since the data is already in the cloud. This assumption is practical since processing on very large data sets often take a large amount of time and money, therefore, users often try to minimize the number of re-executions. In most cases, with a certain set of data, they execute analysis processes once only.

As the future work, we plan to extent SPBD to handle more than one input file, improve the linear assumption (employ piecewise regression for example), and test it with more applications or, better, with a benchmark for large data. Another interesting exploration is to apply GridFTP or other state-of-the-art data transmission protocols instead of TCP.

CHAPTER 8

HIGH EFFICIENT COMPUTING

After identify the solution (using cloud and MapReduce), taming fundamental challenges in heterogeneity and data movement of the system that employs the solution, in this chapter, we would like to target efficiency improvement. Many previous efforts focus solely on improving the utilization of data centers. We argue that even highly utilized systems can still waste the resource because of failures and maintenance. Therefore, in this chapter, we call on a new direction of research on waste-free computing. We start with a workable definition of efficiency, then introduce a new efficiency metric called **usage efficiency**, show how to compute it and propose some possible approaches to improve this type of efficiency. We also calculate the metric through the simulation on real failure traces and diverse workflow applications. While the efficiency computed from the traditional method is 100%, it is less than 44% with our new metric which suggests that the current metrics are inadequate to capture the efficiency of a system.

8.1 Introduction

We have witnessed the fast growing of large-scale data centers because of the promise of cloud computing, running either publicly or privately. In this chapter, we argue that computing resources in those facilities are not being used efficiently. They are either *under-utilized* or *doing meaningless work*. Next we will use four examples to support this argument.

Observation 1: Schroeder and Gibson [120] envision that, by 2013, the utilization of a top500 computer (top500.org) may drop to 0% based on their assumption about the growth in number of cores per socket and the checkpoint overhead. This is because the only job that a future system would be busy with is checkpointing due to the fact that the number of failures in future computers would increase proportionally to the number of cores.

Observation 2: In a typical industrial data center, the average utilization is only 20 to 30 percent [29, 34] (due to the QoS guarantee for example). However, such low utilized or idle servers still draw 60% of the peak power [29, 83, 56]. The average CPU utilization of 5000 Google servers in 6 months is in the 10-50% range which is in the most energy inefficient region [27].

Observation 3: In systems that offer infrastructure as a service (IaaS), the virtualization may lead to low utilization too. Basically, virtualization techniques allow us to “cut” a physical machine into isolated virtual machines. The inefficiency comes not only from the overhead of the splitting process itself but also from the mismatching between virtual and real resources especially in heterogeneous systems. For example, an old machine with only 1.5GHz processor can only offer one EC2-like standard instance(1-1.2GHz) and the rest is wasted.

Observation 4: Together with the rapid penetration of multicore architectures, the utilization of these cores becomes a big issue. It’s quite common to see that one core is very busy while the other ones are pretty idle. For example, when executing the serial part of a program, only one core is needed and others are idle [66]. Furthermore, the issues of maintaining consistency, synchronization, communication, scheduling between many cores might introduce considerable overhead too.

Based on these observations, we categorize the wasted resources into two types: *wasted under-utilized resource* and *wasted utilized resource*. The first type, *wasted under-utilized resource*, is the difference between the available (physical) resources and the consumed resources. It is caused by the low utilization as in the “Observations 2, 3 and 4”. It is used to answer the question: *how many percentages of the available resources are utilized?* The second type, *wasted utilized resource*, is the difference between the consumed resources and the *useful resources* (spent on users’ jobs). It is often caused by the administrative or maintenance tasks as in the “Observation 1 and 4”. It is used to answer the question: *Among utilized resources, how many percentages are spent on doing useful work?*

In addition, the need for resources such as energy is *substantially increasing* as the time of utility computing is approaching. Most data, applications and computation are moving into Cloud. This trend will obviously increase the need of more powerful data centers which are energy hungry. Silicon Valley Leadership Group has also forecasted this increasing trend in the data center energy use based on the Environmental Protection Agency (EPA) reported earlier in 2007 [18, 55].

As a result, using resource *efficiently* or *minimizing the waste* is becoming a very important issue. Minimizing the wasted under-utilized resources, i.e. maximizing the utilization of resources, has been a hot topic in both academic and industrials, such as data center designs and cloud vendors, in order to improve their sustainability. In this chapter, we argue that improving the resource utilization is useful, however, is not enough. We also need to reduce the resource wasted for other purposes such as failures and maintenance (wasted utilized resources). For example, if during its execution, one critical task fails resulting the start-over of the whole job, the resources have been used before the failure are wasted. We envision that those types of resource waste, which are the root of the lower efficiency of computing, should be eliminated significantly. Therefore we introduce **waste-free computing** which considers the efficiency from different angles.

8.2 Efficiency Computation

Essentially, efficiency is a multi-objective problem. Unlike the traditional systems dealing with single-objective optimization problems such as minimizing the execution time, minimizing power consumption or maximizing throughput, etc, efficient systems try to "do more with less". It is obvious that there is a trade-off between objectives. The trade-off between benefits and cost is encapsulated in one metric called efficiency. In this section, we will go over some widely used formulas to calculate the efficiency before deriving our own formula.

Generally, efficiency is defined as follows

$$Efficiency = \frac{Return}{Cost} \tag{8.1}$$

The $Return$ can be profit, the number of work done, improvement of performance (execution time), throughput, bandwidth, availability, queries per second etc. The $Cost$ is what we spend to obtain the $Return$ such as money, energy consumption, CPU time, storage space, etc.

Depending on specific $Return$ and $Cost$, there are different types of efficiency. Let's consider some examples:

Example 1: in term of CPU time, the CPU efficiency is defined as the CPU time spent on doing useful work, divided by the total CPU time. Assume T is the processing time for each process before context switch could occur and S is the overhead for each content switch. For round robin scheduling with quantum Q seconds:

$$\begin{aligned}\text{CPU Efficiency} &= \frac{T}{T+S} \text{ if } Q > T \\ &= \frac{T}{T+S\times\frac{T}{Q}} \text{ if } S < Q < T\end{aligned} \tag{8.2}$$

Example 2: in term of energy, there are many levels of efficiency ranging from chip to data center. At processor level, CPU efficiency is defined as the performance (MIPS) per joule or watt [113]. At storage level, storage efficiency is define as number of queries per second (QPS) per joule or watt [131].

At datacenter level, recent work [42, 48] has defined two widely used metrics (Power Usage Effectiveness (PUE) and Data Center Infrastructure Efficiency(DCiE)) to measure the data center infrastructure efficiency and compare the energy efficiency between data centers.

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} \tag{8.3}$$

$$DCiE = \frac{\text{IT Equipment Power}}{\text{Total Facility Power}} \times 100\% \tag{8.4}$$

Arguing that PUE and DCiE "do not allow for monitoring the effective use of the power supplied – just the differences between power supplied and power consumed," Gartner [37, 106] proposed a new metric called Power to Performance Effectiveness (PPE).

$$PPE = \frac{\text{Useful Work}}{\text{Total Facility Power}} \tag{8.5}$$

Moreover, there is work that considers the combination of efficiency at different levels. For example, in [27], Barroso and Hölzle factorized the Green Grid's Datacenter Performance Efficiency (DCPE) into 3 components to capture different types of energy efficiency such as facility, server energy conversion, and electronic components (from large scale to small scale). They defined energy efficiency metric as (the detail computation can be found in [48, 27, 42])

$$\begin{aligned}\text{Energy Efficiency} &= \frac{Computation}{\text{Total Energy}} \\ &= \left(\frac{1}{PUE}\right) \times \left(\frac{1}{SPUE}\right) \times \\ &\left(\frac{Computation}{\text{Total Energy to Electronic Comp.}}\right)\end{aligned} \tag{8.6}$$

in which SPUE is the server power usage efficiency.

Anderson and Tucek also called for concern about the efficiency especially in data intensive supper computing (DISC) in [23]. In their work, using the sort benchmark, they derived different forms of efficiency such as compute, storage, IO, memory, programmer, management, energy and cost efficiency. Although sharing the same interest, our work is different. We focus on the inefficiency caused by fault-tolerant mechanisms in executing workflows while their aim is to prove that existing DISC system is inefficiency.

The inefficiency is caused by the waste. Where does waste come from? In the first example, the waste is in CPU time. It comes from the context switch time and the scheduling algorithm.

In the second example, the waste is in energy. It comes from unsuitable, inefficient design, architecture, cooling, power conversion and dispensation, etc.

Apart from the previous types of efficiency, we introduce another type of efficiency called **usage efficiency**. With this type, failure is the cause of inefficiency and resource waste. For example, if users submit a job to the system, and it is assigned to unreliable nodes. If some of the nodes fail while executing their job (which is very likely with unreliable nodes), all resources consumed or cost spent on execution before failure are useless. This not only reduces the performance (increase the total execution time) but also increases the total cost. Even worse, this can also affect the schedule of other jobs. Therefore, we introduce a new efficiency metric as the following.

$$\text{Usage efficiency} = \frac{\text{Required Resource}}{\text{Actual Resource Used}} \tag{8.7}$$

The objective is to minimize the number of job re-execution or, more precisely, reduce the probability to execute the job again.

8.3 Usage Efficiency Computation

In this section, we are going to compute the proposed usage efficiency. First we need to identify clearly what are `Required Resource` and `Actual Resource Used`.

In our case, the `Required Resource` is the total execution time of a job running on absolute minimal possible resources without failure and the `Actual Resource Used` is the actual execution time. The execution time is considered as a resource. The more time the job is executed, the more resource it consumes. In the perfect system, i.e. there is no failure, this type of efficiency has value of 1. The general equation (8.7) is rewritten as

$$\text{Usage Efficiency} = \frac{\text{Total exec time w/o failure}}{\text{Real total exec time}} \tag{8.8}$$

In the problem, a `job` is composed of k `tasks`. A job is executed on N processing

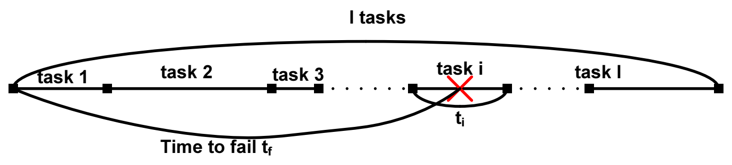


Figure 8.1: The real execution time of a job.

instances (PI) which are either virtual or physical machines.

Let $P_i, (i = 1..N)$ be the probability that the i^{th} PI fails during the job execution time.

Each task has execution time t_j $(j = 1..k)$

For simplicity, we assume that $N > k$ and each PI executes one thread. Therefore, if the PI fails, so does the task running on it. Also, we assume that the failure is of the type fail stop only.

For a job, if one task of a job fails, do we need to re-execute the whole job from the beginning, part of the job or just that task only? Can we reuse the results of the successful, completed tasks? This essentially depends on the dependencies among tasks. With the extremely scale map-reduce like jobs [50], if one task fails, the system only needs to re-execute it. Unfortunately, not all applications fall into such type. Many of them follow work-flow model and there are tasks that are correlated. In such model, we may need to replicate the execution of a task to mask the failure or store the intermediate data to avoid re-execution of the job from the beginning.

Assuming that we know the “critical execution path” of the job which the tasks on it decide the total execution time of the job (Figure 8.1). If any of them is delayed, so does the job. We consider two cases if any of such tasks fail: (1) the execution starts over from the beginning; (2) the execution just restarts the failing tasks. We don't consider replicated execution here. Let l be the length of the path $(1 \leq l \leq k)$. It's also the number of tasks on that path.

8.3.1 Macro-rescheduling: Re-execute the whole job

The probability that all l critical nodes/tasks don't fail during their execution is $\Pi_{i=1}^{l}(1-P_i)$.

Let Y be a random variable denotes the number of times that the job is re-executed. If *task j is re-executed on the same node after failure*, and the *failure probability P_i of the $i^t h$ PI doesn't change* among executions, Y follows Geometric distribution, and therefore has the expectation as the following.

$$E[Y] = \frac{1}{\Pi_{i=1}^{l}(1-P_i)}$$

Without failure, the total execution time of the job is the sum of all tasks on the critical path which is $T = \Sigma_{i=1}^{l} t_i$.

With failure, in this case, the job is start-over from the beginning as shown in Figure 8.1. The probability that at least one critical task fails equals to the probability that the job is re-executed

$$P_{\text{at least 1 fail}} = P_{\text{re-executed}} = 1 - \Pi_{i=1}^{l}(1-P_i)$$

It is noteworthy that although the time to fail is uniformly distributed within t_j for individual critical task j, it is different across tasks.

$\Rightarrow$ the expected time to fail for each execution before the first success of the job is $\frac{\sum_{j=1}^{l} t_j}{2}$.

Let T be the real execution time of the job, and t_{f_n} be the time to fail at the n^{th} execution of the job.

$$T = P_{\text{at least 1 fail}} \times \sum_{n=1}^{Y} t_{f_n} + \Pi_{i=1}^{l}(1-P_i) \times \sum_{j=1}^{l} t_j$$

$\Rightarrow$ the expected execution time of the job is

$$E[T] = P_{\text{at least 1 fail}} \times \left(E[Y] \times \frac{\sum_{j=1}^{l} t_j}{2} \right) + \Pi_{i=1}^{l}(1-P_i) \times \sum_{j=1}^{l} t_j$$

8.3.2 Micro-rescheduling: Re-execute the failing tasks

In this case, the real execution time of the job is simply the sum of that of its individual tasks on the critical path. Let X_i be a random variable denotes the number of times that task i^{th} is re-executed.

$$T = \sum_{i,j=1}^{l} T_{ij} = \sum_{i,j=1}^{l} \left(p_i \times \sum_{n=1}^{X_i} t_{f_n} + (1 - p_i) \times t_j \right)$$

and we have

$$E[T] \quad = \quad \sum_{i,j=1}^{l} \left(p_i \times \left(E[X_i] \times \frac{t_j}{2} \right) + (1 - p_i) \times t_j \right)$$

The usage efficiency for both rescheduling schemes is

$$\text{Usage Efficiency} = \sum_{j=1}^{l} t_j / E[T]$$

For each job, t_j are fixed, but p_i vary among machines, and their values even change over the time. These p_i can be obtained by failure prediction algorithms like [116, 90]. It is noteworthy that we made two very strong assumptions about the re-execution mechanism and the failure probability. In reality, when a task fails, the scheduler can assign another PI, which has a different failure probability, to take care of it. Also, even the failure probability of the same PI may change over the time. Relaxing these assumptions certainly complicates the above computation.

8.4 Usage Efficiency with Real Traces

After showing the computation of the usage efficiency in theory, in this section, we will use the real failure traces and varied workflows to derive the usage efficiency through simulation.

The failure in the simulation includes 10 two-month traces extracted from the real failure trace in [13]. It's done by randomly selecting 32 nodes in the Cluster 2 and randomly choosing two-month periods between calendar year 2001 and 2002.

We simulated the execution of 50 different random weight, communication, height, width DAGs or workflows (extracted from [68]) on 32 processing instances under 4 configurations: `No Failure`, `Recover1_without_lost`, `Recover1_lost` and `RecoverAll`. In the `No Failure`, we didn't apply failure traces. `Recover1` and `RecoverAll` are equivalent to the micro- and macro-rescheduling, respectively. After finishing its assigned task, while waiting for the new task from the scheduler, a node may fail. In this situation, the "lost" means that all data generated by the node is lost, and that finished task needs to be rescheduled. The "without_lost" means all generated data has been saved to a reliable server, and there's no need to reschedule the task once it's done.

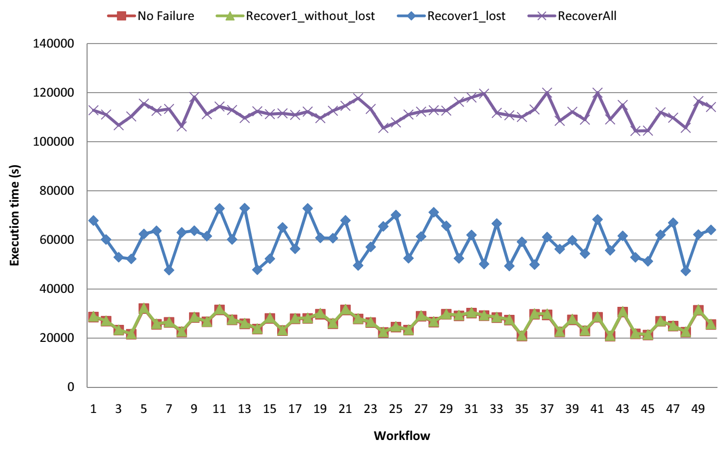


Figure 8.2: The performance of 50 workflows on 32 nodes.

Figure 8.2 shows the execution time of each workflow under 4 configurations. We only use the average of 10 failure traces in the figure for clarity. It's easy to see that the macro-rescheduling and the "lost" configurations take more time to run than the micro-rescheduling and the "without_lost" configurations. The differences between `Recover1_without_lost` and `No Failure` are negligible in comparison to the execution time of the workflow. However, they are indeed different. On average, the `Recover1_without_lost` is about 221 seconds slower than the `No Failure`.

From these results, by averaging all 50 workflows we derived that the usage efficiency of this system is 44% and 23% for the micro and macro-rescheduling, respectively. Actually, in this computation, we consider the execution time of the "No Failure" configuration as the "Required Resource". This value is different if it is computed barely from the graph of the workflow. The value computed from the graph (the critical execution path) doesn't account for the the dependency between tasks. Therefore, our computation isn't affected by the scheduler.

If we consider that the idle time spending on waiting for tasks from the scheduler is also useful (which is true in the Cloud since we have to pay for it even if we don't use it), the efficiency of the system is 100% even with failures since a node is always either busy executing a task or waiting for a task. However, with our computation it's less than 44%.

8.5 Improving Usage Efficiency in Hostile Environments

Failure is an important reason of inefficiency. System designers often apply fault-tolerant techniques such as checkpointing and temporal or spatial redundancy. However, such techniques are often the root of resource waste. Depending on the model of reaction to failures, there are different approaches to improve the efficiency.

With spatial redundancy approach, it wastes the resources in executing replicas. Hence the problem of this approach is to identify suitable number of replicas for each task [32] and their execution locations.

With temporal redundancy approach, the execution is restarted from the beginning in case of failure (macro-rescheduling). One way to improve usage efficiency is to minimize the actual execution time of the job or reduce the number of its re-execution. This can be done by choosing appropriate nodes to assign tasks [136, 96].

With checkpointing approach, the execution is restarted from the checkpoint in case of failure. It potentially creates wasted utilized resources as in the "Observation 1". To address this, one can compress checkpoints or employ process pairs as suggested in [120] or *leverage the non-volatile memory* such as phase change memory (PCM) to avoid the checkpointing overhead.

8.6 Summary and Discussion

Nowadays, we are consuming a great deal of resources in computing, especially in data centers and enterprise environments. However, we observed that many of them are wasted because of a variety of reasons. Therefore, improving the resource usage efficiency or reducing the waste has been receiving increasing attention recently.

There are many different reasons of waste. Among them, failure is our main focus in this work. As the scale of the system increases, failures are common rather than exceptional events. We argued that even highly utilized system can waste the resource since it may be busy with doing useless work only.

In this chapter, we call on a new direction of research on waste-free computing. We start with a workable definition of efficiency, then introduce a new efficiency metric called usage efficiency, and propose some possible approaches to improve this type of efficiency.

Waste-free computing is different from the conventional wisdom of computing such as high performance, high productivity and high throughput computing, although the techniques developed for waste-free computing might help to improve the productivity and throughput of the systems. High performance computing usually focuses on improving the performance of one successful execution, neglecting the time and resources wasted on those unsuccessful execution, which is the target of waste-free computing. For the high productivity computing or high throughput computing, usually the goal is to produce as many results as possible, while waste-free computing focuses on reducing the resource waste, even when the productivity of the system is low.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In summary, we have successfully achieved all suggested research goals. Many solutions have been developed to efficiently manage and process/analyze the big data in genomics related research.

1. In studying the benefit of applying Cloud Computing in Genomic research, we developed CloudAligner - a fast and full-featured MapReduce based tool to align sequences onto chromosomes. Not only does it outperform the traditional sequential tool, its performance gain over another cloud-based counterpart is also significant.

2. In studying how heterogeneity affects the performance/effectiveness of the system and how to tame it, we developed many tools and solutions such as OPERA and DiR. The side effects of these are the improvement in resource usage and performance.

3. In analyzing the storage and processing requirements of particular bioinformatics projects including data formats, data access pattern, security and business models, we have submitted an internal proposal called GenDB, and it were funded.

4. In reducing/minimizing the data transmission when executing analysis jobs inside the cloud, as pointed earlier in the corresponding discussion, we observed that the current MapReduce in the cloud model such as the Amazon EMR wastes a lot of resource on data movement inside the data center. We have developed a distributed cache system and showed that it is able to reduce the traffic of data between computing servers and storage servers in the cloud.

5. In analyzing, modeling, designing and implementing a tool enabling domain scientists to upload data and run analysis easily and securely in the Cloud with minimal respond time,

we have successfully developed an easy-to-use software tool called SPBD that is used by the domain experts to submit jobs and (big) data to the system. The tool takes care of data transferring, processing (with MapReduce framework) and sending the results back with the smallest respond time. The results shows that SPBD can improve the respond time of some popular software with large size input data significantly. If the previous target is about the internal data movement, this one aims not only at the external data movement but the computing time also. Together they provide a complete solution to deal with data movement and processing overhead in the considering system.

6. In studying how to improve the performance, energy consumption and effectiveness we have defined a new method to compute efficiency, showed how to compute it through the simulation on real failure traces and diverse workflow applications, and proposed some possible approaches to improve this type of efficiency.

However, in this work, although we considered the problems at different layers of the system stack, we did not put all solutions together and test them as the whole system. From our collaboration with the two bioinformatic research groups at Wayne State University, we recognized that MapReduce automatic code conversion is a promising direction to follow.

In the first group, we use MySQl to store sequenced data generated by the Illumina GAII sequencer. Then we issued some SQL statements to query information. The database either did not respond or took some days to process the queries that join many tables. This motivated us to find another solution that is more efficient to handle large data.

In the second group, they develop a software in the metagenomics area to categorize the sequences recovered from the environmental samples. However, the software cannot handle very large input data. With an 8GB of memory and 6 cores CPU machine, the software can only process data less than 1GB of input data. With this limitation, there is no way for their tool to be used in reality. It forces us to find another more scalable solution.

We concluded that the metagenomics software, as well as many other existing tools in genomics, is developed under the assumption that all data is available locally, which will limit its applicability from two perspectives: one is the capacity (memory,HDD), another is the speed (not parallel). The appropriate approach for this direction could be to manually rewrite the software in the context of cloud and mapreduce and evaluate the new code to see if they are scalable than the original one. If it is, we should then look for an automatic conversion solution to apply to other similar tools.

BIBLIOGRAPHY

[1] http://dnasequencingtechnologies.com/second-and-third-generation%20sequencing.html.

[2] http://www.csm.ornl.gov/pvm/pvm_home.html.

[3] Amazon ec2 network and s3 performance. http://www.cloudiquity.com/2009/01/amazon-ec2-network-and-s3-performance/.

[4] File system layout. http://open.eucalyptus.com/learn/InstallingECC.

[5] http://blog.cloudharmony.com/2011/04/unofficial-ec2-outage-postmortem-sky-is.html.

[6] http://en.wikipedia.org.

[7] http://ganglia.info/.

[8] http://wiki.apache.org/hadoop/.

[9] http://www.ebay.com/.

[10] http://www.gartner.com.

[11] http://www.illumina.com/systems/hiseq_systems.ilmn.

[12] http://www.novocraft.com/main/index.php.

[13] Los alamos national laboratory. operational data to support and enable computer science research.

[14] Pentaho for big data.

[15] Virtualization basics. http://www.vmware.com.

[16] wikimedia dump.

[17] www.wattsupmeters.com.

[18] Data center energy forecast, 7 2008.

[19] A. Jsang A. Whitby and J. Indulska. Filtering out unfair ratings in bayesian reputation systems. In *The Icfain Journal of Management Research*, volume 4, pages 48–64, February 2003.

[20] Anurag Agarwal and Anand Mitra. Introduction to server virtualization. *http://punetech.com*, 2008.

[21] Bryce Allen, John Bresnahan, Lisa Childers, Ian Foster, Gopi Kandaswamy, Raj Kettimuthu, Jack Kordas, Mike Link, Stuart Martin, Karl Pickett, and Steven Tuecke. Software as a service for data scientists. *Commun. ACM*, 55(2):81–88, February 2012.

[22] Beulah Kurian Alunkal, Ivana Veljkovic, Gregor Von Laszewski, and Kaizar Amin. Reputation-based grid resource selection. In *Workshop on Adaptive Grid Middleware*, page 28, 2003.

[23] Eric Anderson and Joseph Tucek. Efciency matters! In *HotStorage*, 2009.

[24] Samuel Angiuoli, Malcolm Matalka, Aaron Gussman, Kevin Galens, Mahesh Vangala, David Riley, Cesar Arze, James White, Owen White, and W Florian Fricke. Clovr: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC Bioinformatics*, 12(1):356, 2011.

[25] F. Azzedin and M. Maheswaran. Evolving and managing trust in grid computing systems. In *in Proceedings of IEEE Canadian Conference on Electrical & Computer Engineering*, pages 1424–1429, May 2002.

[26] Luiz A. Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, volume Lecture #6.

[27] Luiz A. Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, volume Lecture #6.

[28] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[29] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[30] J. Batley and D. Edwards. Genome sequence data: management, storage, and visualization. *Biotechniques*, 46(5):333–336, 2009.

[31] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proc. of NSDI'04*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[32] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proc. of NSDI'04*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[33] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case for power management in web servers. pages 261–289, 2002.

[34] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case for power management in web servers. pages 261–289, 2002.

[35] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. SIGMETRICS*, 2000.

[36] R. Buyya, D. Abramson, and J. Giddy. Nimrod-G: An architecture for a resource management and scheduling system in a global computational grid. In *HPCAsia 2000*, May 2000.

[37] David Cappuccio. Data center efficiency beyond pue and dcie.

[38] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 390–399. IEEE, 2011.

[39] Ann L. Chervenak and Robert Schuler. A data placement service for petascale applications. In *PDSW '07: Proceedings of the 2nd international workshop on Petascale data storage*, pages 63–68, New York, NY, USA, 2007. ACM.

[40] Ann L. Chervenak, Robert Schuler, Matei Ripeanu, Muhammad Ali Amer, Shishir Bharathi, Ian Foster, Adriana Iamnitchi, and Carl Kesselman. The globus replica location service: Design and experience. *IEEE Transactions on Parallel and Distributed Systems*, 99(1), 2008.

[41] John Pfleuger Christian Belady, Andy Rawson and Tahir Cader. Green grid data center power efficiency metrics: Pue and dcie. www.thegreengrid.org, 2008.

[42] John Pfleuger Christian Belady, Andy Rawson and Tahir Cader. Green grid data center power efficiency metrics: Pue and dcie. www.thegreengrid.org, 2008.

[43] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica mainte-

nance for distributed storage systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[44] Jeffrey Clark. Us government to consolidate data centers? *The Data Center Journal*, 2010.

[45] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):177–190, 2002.

[46] Bled Electronic Commerce, Audun Jøsang, and Roslan Ismail. The beta reputation system. In *In Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.

[47] Jud Cooley Mark Monroe Pam Lembke Dan Azevedo, Jon Haas. How to measure and report pue and dcie. www.thegreengrid.org, 2008.

[48] Jud Cooley Mark Monroe Pam Lembke Dan Azevedo, Jon Haas. How to measure and report pue and dcie. www.thegreengrid.org, 2008.

[49] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[50] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[51] Shyamala Doraimani. *Filecules: A New Granularity for Resource Management in Grids*. Master thesis in Computer Science, University of South Florida, 2007.

[52] J.R. Douceur and R.P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, pages 4–13, 2001.

[53] Tom White Doug Cutting and Steve Loughran. whirr procject.

[54] EPA. Epa report to congress on server and data center energy efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[55] EPA. Epa report to congress on server and data center energy efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[56] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[57] Justin Fielding. The benefits of virtualisation, 2006. `http://blogs.techrepublic.com.com`.

[58] Paul Flicek. Challenges for the data management and analysis of large-scale human genome sequencing, 2010.

[59] Cuenca-Acunam FM, R.P. Martin, and T.D. Nguyen. Autonomous replication for high availability in unstructured p2p systems. *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 99–108, Oct. 2003.

[60] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Proc. of the 10th International Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.

[61] Robert P. Goldberg. Survey of virtual machine research. *IEEE Computer*, pages 34–45, 1974.

[62] Jonathan Corbet Greg Kroah-Hartman and Amanda McPherson. Linux kernel development. *www.linuxfoundation.org*, 2008.

[63] Yunhong Gu and Robert L. Grossman. Sector and sphere: the design and implementation of a high-performance data cloud. *Philosophical Transactions of the Royal Society A: Mathematical,Physical and Engineering Sciences*, 367(1897):2429–2445, 2009.

[64] Troy D. Hanson. uthash: a hash table for c structure.

[65] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM.

[66] Mark Hill. Amdahl's law in the multicore era, Jan 2010.

[67] N. Homer, B. Merriman, and S.F. Nelson. BFAST: an alignment tool for large scale genome resequencing. *PLoS One*, 4(11):e7767, 2009.

[68] U. Honig and W. Schiffmann. A comprehensive test bench for the evaluation of scheduling heuristics. In *Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS04), Cambridge, USA*, 2004.

[69] W. Huang, M.J. Koop, and D.K. Panda. Efficient one-copy mpi shared memory communication in virtual machines. In *Cluster Computing, 2008 IEEE International Conference on*, pages 107–115. IEEE, 2008.

[70] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 125–134, New York, NY, USA, 2006. ACM.

[71] Shadi Ibrahim, Hai Jin, Bin Cheng, Haijun Cao, Song Wu, and Li Qi. Cloudlet: towards mapreduce implementation on virtual machines. In *Proceedings of the 18th ACM in-*

ternational symposium on High performance distributed computing, HPDC ’09, pages 65–66, New York, NY, USA, 2009. ACM.

[72] Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, and Xuanhua Shi. Evaluating mapreduce on virtual machines: The hadoop case. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom ’09, pages 519–528, Berlin, Heidelberg, 2009. Springer-Verlag.

[73] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, March 2007.

[74] Matt Healey Jed Scaramella. Service-based approaches to improving data center thermal and power efficiencies. IDC White Paper, May 2007.

[75] M. A. Jøsang and E.Gray. Analysing topologies of transitive trust. In *Proceedings of the Workshop of Formal Aspects of Security and Trust (FAST)*, September 2003.

[76] Luke Jostins. Basics: Second-generation sequencing, Sept 2010.

[77] S. Kamvar, M. T. Schlosser, and H. Gacia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. of the 12th International World Wide Web Conference (2003)*, May 2003.

[78] Julia Karow. Illumina details hiseq improvements at agbt; early-access users weigh in, Feb 2011.

[79] Evangelos Kotsovinos and Douglas Mcilwraith. replic8: Location-aware data replication for high availability in ubiquitous environments. In *Wired/Wireless Internet Communications*, volume 3510, pages 32–41, Berlin / Heidelberg, 2005. Springer.

[80] Konstantinos Krampis, Tim Booth, Brad Chapman, Bela Tiwari, Mesude Bicak, Dawn Field, and Karen Nelson. Cloud biolinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics*, 13(1):42, 2012.

[81] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[82] Ben Langmead, Michael Schatz, Jimmy Lin, Mihai Pop, and Steven Salzberg. Searching for snps with cloud computing. *Genome Biology*, 10(11):R134, 2009.

[83] C. Lefurgy, Xiaorui Wang, and M. Ware. Server-level power control. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 4–4, June 2007.

[84] H Li. Maq: Mapping and assembly with qualities. *Version 0.6.3*, 2008.

[85] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589, 2010.

[86] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078, 2009.

[87] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrowswheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[88] Yanen Li and Sheng Zhong. Seqmapreduce: software and web service for accelerating sequence mapping. In *Critical Assessment of Massive Data Anaysis (CAMDA) 2009*, 2009.

[89] Yawei Li and Zhiling Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:531–538, 2006.

[90] Yawei Li and Zhiling Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:531–538, 2006.

[91] Qiao Lian, Wei Chen, and Zheng Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 187–196, Washington, DC, USA, 2005. IEEE Computer Society.

[92] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, Jose Moreira, and Manish Gupta. Filtering failure logs for a bluegene/l prototype. volume 0, pages 476–485, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[93] Z. Liang and W. Shi. Analysis of recommendations on trust inference in the open environment. Technical Report MIST-TR-2005-002, Department of Computer Science, Wayne State University, Feb. 2005.

[94] Z. Liang and W. Shi. Enforcing cooperative resource sharing in untrusted peer-to-peer environment. *ACM Journal of Mobile Networks and Applications (MONET) special issue on Non-cooperative Wireless networking and computing*, 10(6):771–783, Dec. 2005.

[95] Zhengqiang Liang and Weisong Shi. Analysis of ratings on trust inference in open environments. *Elsevier Performance Evaluation*, 65(2):99–128, Feb. 2008.

[96] Zhengqiang Liang and Weisong Shi. A reputation-driven scheduler for autonomic and sustainable resource sharing in grid computing. *J. Parallel Distrib. Comput.*, 70(2):111–125, 2010.

[97] J. Liu, W. Huang, B. Abali, and D.K. Panda. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the USENIX Annual Technical Conference*, pages 3–3, 2006.

[98] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. Greencloud: a new architecture for green data center. In *ICAC-INDST '09: Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, pages 29–38, New York, NY, USA, 2009. ACM.

[99] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[100] Daniel MacArthur. Illumina announces new arrival in the sequencing arms race, 2010.

[101] SALIM HARIRI Manish Parashar. Autonomic computing: An overview. In *In Proceedings of the Intl Workshop on Unconventional Programming Paradigms (UPP'04) (2005)*, pages 257–269. Springer Berlin-Heidelberg, 2005.

[102] H.M. Monti, A.R. Butt, and S.S. Vazhkudai. Catch: A cloud-based adaptive data transfer service for hpc. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1242 –1253, may 2011.

[103] Tung Nguyen, Weisong Shi, and Douglas Ruden. Cloudaligner: A fast and full-featured mapreduce based tool for sequence mapping. *BMC Research Notes*, 4(1):171, 2011.

[104] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:124–131, 2009.

[105] Giwon On, Jens Schmitt, and Ralf Steinmetz. The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 57, Washington, DC, USA, 2003. IEEE Computer Society.

[106] Raymond Paquet. Technology trends you cant afford to ignore. Gartner Webinar, December 2009.

[107] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[108] Xiaohong Qiu, Jaliya Ekanayake, Scott Beason, Thilina Gunarathne, Geoffrey Fox, Roger Barga, and Dennis Gannon. Cloud technologies for bioinformatics applications. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 6:1–6:10, New York, NY, USA, 2009. ACM.

[109] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. *High-Performance Distributed Computing, International Symposium on*, 0:140, 1998.

[110] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content addressable network. In *Proc. of ACM SIGCOMM'01*, 2001.

[111] Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, and Saurabh Bagchi. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, 5:173–195, 2007. 10.1007/s10723-007-9077-5.

[112] Daniel C. Richter, Felix Ott, Alexander F. Auch, Ramona Schmid, and Daniel H. Huson. Metasim - a sequencing simulator for genomics and metagenomics. *PLoS ONE*, 3(10):e3373, 10 2008.

[113] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, Christos Kozyrakis, and Justin Meza. Models and metrics to enable energy-efficiency optimizations. *Computer*, 40:39–48, 2007.

[114] U. Röhm and J.A. Blakeley. Data management for high-throughput genomics. In *Proc. CIDR*, 2009.

[115] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, 2001.

[116] F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, April 2006.

[117] M.C. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363, 2009.

[118] James Schnable. Even faster sequencing, 2010.

[119] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an mttf of 1,000000 hours mean to you? In *Proc. of the 5th USENIX Conf. On File and Storage Technologies*, pages 1–9, February 2007.

[120] B. Schroeder and G.A. Gibson. Understanding failures in petascale computers. 78:012022, 2007.

[121] S. Seo, I. Jang, K. Woo, I. Kim, J.S. Kim, and S. Maeng. Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–8. IEEE, 2009.

[122] Jana Technology Services. Differences between s3 and ebs.

[123] J. Shafer, S. Rixner, and A.L. Cox. The hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 122–133. IEEE, 2010.

[124] Amit Singh. An introduction to virtualization, 2005. `http://www.kernelthread.com`.

[125] A.D. Smith, W.Y. Chung, E. Hodges, J. Kendall, G. Hannon, J. Hicks, Z. Xuan, and M.Q. Zhang. Updates to the rmap short-read mapping software. *Bioinformatics*, 25(21):2841, 2009.

[126] Andrew Smith, Zhenyu Xuan, and Michael Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9(1):128, 2008.

[127] Jason Sonnek, Abhishek Chandra, and Jon Weissman. Adaptive reputation-based scheduling on unreliable distributed infrastructures. *IEEE Trans. Parallel Distrib. Syst.*, 18(11):1551–1564, 2007.

[128] J.D. Sonnek and J.B. Weissman. A quantitative comparison of reputation systems in the grid. *Grid Computing, IEEE/ACM International Workshop on*, 0:242–249, 2005.

[129] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM'2001*, 2001.

[130] Thorsten. Network performance within amazon ec2 and to amazon s3.

[131] Vijay Vasudevan, Jason Franklin, David Andersen, Amar Phanishayee, Lawrence Tan, Michael Kaminsky, and Iulian Moraru. FAWNdamentally power-efficient clusters. In *Proc. HotOS XII*, Monte Verita, Switzerland, May 2009.

[132] An-I A. Wang, Peter Reiher, and Geoff Kuenning. Introducing permuted states for analyzing conflict rates in optimistic replication. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 376–377, New York, NY, USA, 2005. ACM.

[133] John C. Wooley, Adam Godzik, and Iddo Friedberg. A primer on metagenomics. *PLoS Comput Biol*, 6(2):e1000667, 02 2010.

[134] Sandeep Yadav. New sequencing technology.

[135] Richard M. Yoo, Anthony Romano, and Christos Kozyrakis. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. *IEEE Workload Characterization Symposium*, 0:198–207, 2009.

[136] Chenjia Yu, Zhifeng Wang and Weisong Shi. Flaw: Failure-aware workflow scheduling in high performance computing environments. Technical report mist-tr-2007-010, Waye State University, Nov 2007.

[137] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, Apr*, pages 2009–55, 2009.

[138] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys 2010*. EuroSys, 2010.

[139] Matei Zaharia, Andrew Konwinski, Anthony Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th symposium on Operating systems design and implementation*, pages 29–42, San Diego, CA, 12/2008 2008. USENIX Association, OSDI.

[140] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Wang, and Shengzhong Feng. Accelerating mapreduce with distributed memory cache. In *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, pages 472 –478, dec. 2009.

[141] Chenjia Wang Zhifeng Yu and Weisong Shi. Failure-aware workflow scheduling in cluster environments. *Cluster Computing Journal*, 13:421–434, 2010.

[142] Ming Zhong, Kai Shen, and Joel Seiferas. Replication degree customization for high availability. *SIGOPS Oper. Syst. Rev.*, 42(4):55–68, 2008.

ABSTRACT

SYSTEMS SUPPORT FOR GENOMICS COMPUTING IN CLOUD ENVIRONMENTS

by

TUNG NGUYEN

December 2012

Advisor: Dr. Weisong Shi

Major: Computer Science

Degree: Doctor of Philosophy

Genomics research has enormous applications in many areas such as health care, forensic, agriculture, etc. Most recent achievements in this field come from the availability of the unprecedented genomic data. However, new sequencing technologies in genomics keep producing data at a faster pace resulting a very huge amount of data. This poses great challenges on how to store, manage, process and analyze the data efficiently. To deal with these, genomics research groups often equip themselves with a small scale server room composed of high storage capacity and computing ability machines. This solution is not only costly, unscalable but also inefficient. A better solution would be the Cloud Computing with its elasticity and pay-as-you-go economic model. Nevertheless, Cloud Computing only provides the potential infrastructure solution. To address the high-throughput processing challenges, we need to have a suitable programming model. The fundamental idea is to process data in parallel. In existing models, MapReduce appears to be the best candidate because of its extremely scalability.

In this work, we plan to develop a domain specific style system to support data management and analysis in genomics using Cloud Computing and MapReduce. Starting from the application layer, we developed a fundamental alignment tool called CloudAligner based on the MapReduce framework that outperformed its counterparts. After that, we continue seeking

solutions to improve the system at the infrastructure level. Observing that scientists spend too much time on accessing data from low speed archives (tapes), we developed the Distributed Disk Cache (DiSK), and it was covered in a Master thesis. Another challenge is to enable the system to support differentiated services which are prevalent in Cloud Computing. To address this, we proposed a Differentiated Replication (DiR) mechanism allowing data to be inserted and retrieved with different availability. Another problem that greatly reduces the performance of the system is the heterogeneity of the Cloud. To tame it, we created an Open Reputation model called Opera. It employs vectors to record the behaviors (reputations) of nodes from different aspects. We modified the Hadoop MapReduce scheduler to make use of this information. The results proved that under heterogeneous environments, our system is better than the original Hadoop in terms of job execution time, number of failed/killed tasks, and energy consumption. The last challenge we have dealt with is the data movement since the data in our targeted domain (genomics) is extremely large and is generated with exponential rate. We divided the issue into two categories: internal and external movement. We have successfully developed a cached system to minimize the internal data movement and an easy-to-use tool called SPBD to handle external data movement with minimal respond time.

AUTOBIOGRAPHICAL STATEMENT

TUNG NGUYEN

Tung Nguyen joined PhD program at Wayne State University in 2007. He received his Bachelor and Master degrees in Computer Science and Engineering from HoChiMinh City University of Technology, Vietnam in 2001 and 2006, respectively. His research interests include Green Computing, Cloud Computing, Data Intensive Computing, and application of Cloud Computing to life science. He has published several papers in workshops, conferences and journal in both Computer Science and Bioinformatics such as OSDI, NPC, SUSCOM, BMC, Frontiers Genetics, etc. He has also served as a peer reviewer for many conferences such as euro-par, CollaborateCom, etc. More information can be found on his homepage at http://www.cs.wayne.edu/tung/ or with Google.