

Louisiana State University

LSU Scholarly Repository

LSU Doctoral Dissertations

Graduate School

6-23-2018

A Study of Scalability and Cost-effectiveness of Large-scale Scientific Applications over Heterogeneous Computing Environment

Arghya K. Das

Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://repository.lsu.edu/gradschool_dissertations



Part of the [Bioinformatics Commons](#), [Computational Engineering Commons](#), [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), [Hardware Systems Commons](#), [Systems and Communications Commons](#), and the [Systems and Integrative Engineering Commons](#)

Recommended Citation

Das, Arghya K., "A Study of Scalability and Cost-effectiveness of Large-scale Scientific Applications over Heterogeneous Computing Environment" (2018). *LSU Doctoral Dissertations*. 4651.
https://repository.lsu.edu/gradschool_dissertations/4651

This Dissertation is brought to you for free and open access by the Graduate School at LSU Scholarly Repository. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Scholarly Repository. For more information, please contact gradetd@lsu.edu.

A STUDY OF SCALABILITY AND COST-EFFECTIVENESS OF
LARGE-SCALE SCIENTIFIC APPLICATIONS OVER HETEROGENEOUS
COMPUTING ENVIRONMENT

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by

Arghya Kusum Das

B.Tech, West Bengal University of Technology, 2008

August 2018

Acknowledgments

I would like to thank my advisor Dr. Seung-Jong Park, co-advisor Dr. Kisung Lee and Dr. Seungwon Yang for guiding me in proper direction throughout my Ph.D. Thanks to Dr. Jianhua Chen for serving as my committee member and provide her valuable opinion in other collaborative projects in our group which enriched the thesis content. I would like to thank Dr. Prosanta Chakrabarty for his time to serve as the Dean's Representative in the committee. Thanks to Dr. Ling Liu (GATECH) for collaborating with us on part of this thesis.

I would also like to thank all my team members Praveen Koppa, Sayan Goswami, Richard Platania, Shayan Shams, Chui Hui Chiu and Dipak Singh for all their help and suggestions. Also, thanks to the LSU-HPC and LONI (Louisiana Optical Network Infrastructure) team for the excellent HPC service throughout the research.

Finally, I would like to thank the Samsung SSD team in S. Korea including Jaeki Hong, Jinki Kim, Jay Seo and Wooseok Chang for their support in evaluating cutting-edge hardware and cluster architectures.

This research is supported in part by the following grants: NIH-P20GM103424, NSF-MRI-1338051, NSF-CC-NIE-1341008, NSF-IBSS-L-1620451 and LA BoR LEQSF (2016-19)-RD-A-08

Table of Contents

ACKNOWLEDGMENTS	ii
LIST OF TABLES	vi
LIST OF FIGURES	viii
ABSTRACT	xii
CHAPTER	
1 INTRODUCTION	1
1.1 Goals	2
1.2 Objective.....	3
1.3 Research Outline	6
2 BACKGROUND	7
2.1 Brief History of Big Data and Big Science	7
2.2 Big Data Challenges in Genomics	8
2.3 Computation and Storage Models for Scientific Big Data	9
2.4 Hardware Technologies for Big Data HPC	14
2.5 Data Sharing Model for Scientific Big Data.....	15
3 PARALLEL SHORT-READ ERROR CORRECTION USING HADOOP	17
3.1 Introduction.....	17
3.2 Related Work	19
3.3 Methodology	21
3.4 Evaluation	28
3.5 Conclusion	33
4 GIRAPH-BASED GENOME ASSEMBLER FOR LARGE-SCALE GENOMES	34
4.1 Introduction.....	34
4.2 Related Work	36
4.3 Methodology	38
4.4 Evaluation	45
4.5 Conclusion	50
5 PARALLEL LONG-READ ERROR CORRECTION WITH HADOOP	51
5.1 Introduction.....	51
5.2 Related Work	53

5.3	Methodology	55
5.4	Evaluation	63
5.5	Conclusion	69
6	EVALUATING DIFFERENT DISTRIBUTED CYBERINFRASTRUCTURE FOR DATA AND COMPUTE INTENSIVE APPLICATIONS.....	71
6.1	Introduction.....	72
6.2	Related Work	74
6.3	Motivation: Issue in Running Big Data Applications on Traditional Supercomputers	76
6.4	Evaluation Methodology	80
6.5	Impact of Different Hardware Component	87
6.6	Impact of Different Hardware-Organizations.....	92
6.7	Conclusion	99
7	A THEORETICAL MODEL FOR COST-BALANCED HPC CLUSTER FOR DATA SCIENCE.....	100
7.1	Introduction.....	100
7.2	Related Work	102
7.3	Background	104
7.4	Proposed Model for System Balance	105
7.5	Experimental Testbeds: Critical Analysis of Architectural Balance.....	112
7.6	Cluster Evaluation Methodology	116
7.7	Results and Discussion.....	118
7.8	Conclusion	123
8	HIGH THROUGHPUT TRANSACTION OF BIG BIOMEDICAL DATA WITH BLOCKCHAIN AND P2P STORAGE	124
8.1	Introduction.....	125
8.2	Related Work	127
8.3	Background	130
8.4	SwarMed Architecture	132
8.5	Evaluation	140
8.6	Operating Cost of SwarMed on Public Network (Internet)	154
8.7	SwarMed and ONC's Interoperability Roadmap	155
8.8	Conclusion	156
9	CONCLUSION AND FUTURE WORK.....	158
	REFERENCES.....	161

APPENDIX: COPYRIGHT AND PERMISSION INFORMATION	172
VITA	180

List of Tables

3.1	Dataset (simulated and real) used to evaluate ParSECH.....	29
3.2	Accuracy of <i>E.coli</i> and Human genome for different coverage with $k = 15$. (using BWA alignment)	29
3.3	Performance on real <i>E. coli</i> genome ($k = 15$).....	31
3.4	Performance on real human genome ($k = 15$).....	32
4.1	Datasets	46
4.2	Accuracy of <i>S. aureus</i>	48
4.3	Accuracy of <i>R. sphaeroides</i>	48
4.4	Accuracy of HCR 14	49
4.5	Assembly of <i>E. coli</i> using $k = 27$	49
4.6	Assembly of Yoruban male genome using $k = 57$	50
5.1	PacBio dataset	64
5.2	Illumina dataset.....	64
5.3	Compute environment	64
5.4	Different types of algorithms: Widest-Path (ParLECH _{WP}) vs Dijkstra’s shortest-path (ParLECH _{SP}) vs greedy algorithm (ParLECH _{Gr}).	66
5.5	ParLECH accuracy: ParLECH is more accurate than LoRDEC both in terms of alignment and gain	67
5.6	Correcting human genome	69
6.1	Experimental testbeds with different configurations	81
6.2	Hardware components of different cluster configurations, and their cost	82
6.3	Moderate-size bumble bee genome assembly.....	86
6.4	Large-size human genome assembly	86

7.1	Notations used in the model and their meaning	106
7.2	Cost of different hardware components	112
7.3	Experimental testbeds	113
7.4	Data size for different benchmark applications	117
7.5	Resources in each cluster architecture used for TeraSort and WordCount	119
7.6	Maximum available resources in each cluster architec- ture (used for large human genome assembly)	122
8.1	Common Data Model (CDM) of health care. All the records are synthesized with computer programs follow- ing this model	141
8.2	Compute Environment	142
8.3	Operating cost of different designs	154

List of Figures

1.1	Cost per genome sequence (Source: NHGRI)	4
2.1	Genome sequencing pipeline	9
2.2	MapReduce model of computation	11
2.3	Bulk Synchronous Parallel (BSP) model of computation	12
2.4	Distributed NoSQL storage model	13
2.5	Blockchain transaction model	16
3.1	ParSECH architecture	23
3.2	Scalability of ParSECH with different data size and different #nodes	31
3.3	ParSECH's scaled out (4cores and 32GB memory/node) performance vs Quake's scaled up (20cores and 1TB memory) performance for large data	33
4.1	De Bruijn graph construction using Hadoop	39
4.2	Initial Compression: Each two supersteps make a round. Dotted lines show the messages	41
4.3	Tip removal: Each two supersteps make a round.. Dotted lines show the messages	43
4.4	Bubble removal: Each two supersteps make a round. Dotted lines show the messages.	45
4.5	Scalability result	48
5.1	Error correction steps	56
5.2	Widest path algorithm has higher probability to produce correct result even when high coverage k -mers are present in the error path	58
5.3	ParLECH's distributed architecture and error correction pipeline	59

5.4	De Bruijn graph construction using Hadoop.	60
5.5	ParLECH scalability	68
6.1	Impact of each individual hardware component on execution time of the assembly pipeline in 15-DN	89
6.2	CPU utilization and I/O wait characteristics in SwatIII-Basic-HDD (1-HDD/node) and SwatIII-Basic-SSD (1-SSD/node)	90
6.3	Comparison of disk-write IOPS (write) on local file system (of each datanode) and I/O throughput for HDFS-write (across all datanodes) for HDD and SSD	91
6.4	Performance comparison among different type of cluster architectures in terms of normalized execution time and performance-to-price	93
6.5	Comparison of different types of cluster architecture for human genome assembly pipeline.	95
6.6	Performance trend using HDD and SSD in Hadoop. SSD shows better performance and scalability in a scaled out environment.....	97
7.1	Change in system's optimum I/O balance (β_{io}^{opt}) and memory balance (β_{mem}^{opt}) as a function of application balance (γ_{io} and γ_{mem}) for different cost balance (δ_{io} and δ_{mem}).	110
7.2	Execution time (normalized to the baseline) of TeraSort and WordCount over different cluster architectures keeping the total cost of each cluster same.....	119
7.3	CPU and I/O characteristics of each node of different cluster architectures for TeraSort and WordCount benchmark	121
7.4	Performance of different cluster for large size human genome assembly (normalized to 128 nodes of SuperMikeII).....	122
8.1	Interoperability model	133
8.2	SwarMed's decentralized architecture	134
8.3	Smart contracts	135

8.4	SwarMed indexing service	139
8.5	Transferring 8Mn patient record over P2P storage and from an HTTP server	144
8.6	Writing 8Mn patient records.....	145
8.7	Swarm and IPFS writer node statistics.....	146
8.8	Blockchain performance	148
8.9	Blockchain+Swarm write scalability	151
8.10	Blockchain+Swarm write scalability	151
8.11	Blockchain+Swarm design alternatives	152
8.12	Execution time for upload 8Mn patient records to Swarm for sharing	154

Abstract

Recent advances in large-scale experimental facilities ushered in an era of data-driven science. These large-scale data increase the opportunity to answer many fundamental questions in basic science. However, these data pose new challenges to the scientific community in terms of their optimal processing and transfer. Consequently, scientists are in dire need of robust high performance computing (HPC) solutions that can scale with terabytes of data.

In this thesis, I address the challenges in three major aspects of scientific big data processing as follows: 1) Developing scalable software and algorithms for data- and compute-intensive scientific applications. 2) Proposing new cluster architectures that these software tools need for good performance. 3) Transferring the big scientific dataset among clusters situated at geographically disparate locations.

In the first part, I develop scalable algorithms to process huge amounts of scientific big data using the power of recent analytic tools such as, Hadoop, Giraph, NoSQL, etc. At a broader level, these algorithms take the advantage of locality-based computing that can scale with increasing amount of data. The thesis mainly addresses the challenges involved in large-scale genome analysis applications such as, genomic error correction and genome assembly which made their way to the forefront of big data challenges recently.

In the second part of the thesis, I perform a systematic benchmark study using the above-mentioned algorithms on different distributed cyberinfrastructures to pinpoint the limitations in a traditional HPC cluster to process big data. Then I propose the solution to those limitations by balancing the I/O bandwidth of the solid state drive (SSD) with the computational speed of high performance CPUs. A theoretical model has been also proposed to help the HPC system designers who are striving for system balance.

In the third part of the thesis, I develop a high throughput architecture for transferring these big scientific datasets among geographically disparate clusters. The architecture leverages the power of Ethereum's Blockchain technology and Swarm's peer-to-peer (P2P) storage technology to transfer the data in secure, tamper-proof fashion. Instead of optimizing the computation in a single cluster, in this part, my major motivation is to foster translational research, and data interoperability in collaboration with multiple institutions.

Chapter 1

Introduction

In the last few years, the large-scale experimental facilities made a significant advancement in their underlying technologies including both hardware and software. As a result, these big machines started producing an unprecedented amount of big data. For example, next-generation DNA sequencing machines such as Illumina Genome Analyzer, HiSeq, etc. in the last decade outpaced Moore's law and started producing multiple terabytes of data with an unprecedented throughput never seen before. The LIGO (Laser Interferometer Gravitational-Wave Observatory) observatory at Livingstone, or the Large Hadron Collider at Geneva also produces terabytes of data which can answer many of the arcane, fundamental questions of basic science if analyzed properly. As a matter of fact, because of the big budgets, big machines and significant advances in technologies, the last decade experienced a data deluge not only in the field of life science and astrophysics, but also in the other domains of science such as social science, environmental science, and chemistry.

These huge amounts of big data, if analyzed properly can answer many obscure and fundamental questions of different domains of science unveiling complex patterns in human behavior or basic particles of the universe. However, the computation on these huge amounts of big data is severely constrained by the traditional model of computation as well as the traditional HPC infrastructure. In fact, with the last few years of experiences, it is well accepted in the researcher and HPC community that the 'traditional supercomputers focused on performing calculations at blazing speeds have fallen behind when it comes to sifting through huge amounts of big data'¹. Furthermore, the sharing of big data among multiple institutes

¹<https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-redesigned-supercomputers-to-solve-big-data-problems>

to advance the translational research pose extra challenges in terms of security and privacy especially when the data is confidential and protected by government regulations.

This thesis is motivated to help both the scientists as well as the HPC system designers by providing a systematic approach for how to manage these data-intensive applications both in terms of a computational model as well as cost-effective, yet high performance cyberinfrastructure. The best practical approach till date was originally proposed by Jim Gray in 2000 (known as Gray's Laws [2] [3]) where he stated, 1) Bring computations to the data, rather than data to the computations, and 2) The solution is in a scale-out architecture. With the increasing data size, the first law, which basically defines the locality-based model of computation utilizing lower network bandwidth, seems to govern the scientific big data analysis for next several years in future. Consequently, a plethora of applications in the HPC domain needs to be redesigned and restructured to fit this model. On the other hand, the second law, which defines the underlying distributed cyberinfrastructure, also needs to be clarified. To this end, the question that is becoming increasingly important is 'how does the next generation HPC cluster for scientific big data analytics should look like'.

1.1 Goals

The thesis identified the following three different areas where the existing HPC solutions are severely challenged by the enormous amount of data.

1. Scalability issues in existing HPC software for optimal processing of big data.
2. Issues in existing cyberinfrastructure for optimal storage and processing of big data.
3. Robustness and scaling issues in real time sharing of big data.

The major goal of this thesis is to address the challenges in each of these pain

areas of scientific big data analysis. In the first part, the thesis addresses the issue of optimal processing of big data by developing scalable software for processing scientific big data such as genomics data using state-of-the-art programming models such as MapReduce, vertex centric graph processing, distributed NoSQL, etc. The second part shows how these software tools developed atop those relatively newer programming models can be benefited using the recent advances in storage technologies such as flash-based storage and SSD. Finally, in the third part, the thesis proposes a high throughput framework to transfer big data among multiple clusters using Blockchain and peer-to-peer storage technology in a secured manner. Thus, in the end, the thesis provides a holistic view of challenges and the solutions in scientific big data processing in three different levels such as, scalable software, high performance hardware, and high throughput transfer.

1.2 Objective

To address the challenges in scientific big data analysis, first, a specific problem area is required to be identified in the domain of big data science. In this thesis, the bioinformatics and biomedical domain have been selected as the major area of focus. The challenges are enormous in terms of processing, storage, and sharing of these datasets. This thesis will attempt to resolve these challenges from the perspective of both software and hardware.

With the recent advances in high throughput DNA sequencing machine and different medical instruments, bioinformatics and biomedical datasets have exceeded terabytes and will continue to grow as the technologies will improve. As shown in Figure 1.1, next-generation sequencing (NGS) outpaced Moore's law and started sequencing at high throughput at substantially lower cost. However, lengths of the reads are small, and the genome is oversampled to cover every nucleotide base position with high confidence. These aspects of NGS technologies make de novo genome analysis a severely data-intensive endeavor. Furthermore, the reads pro-

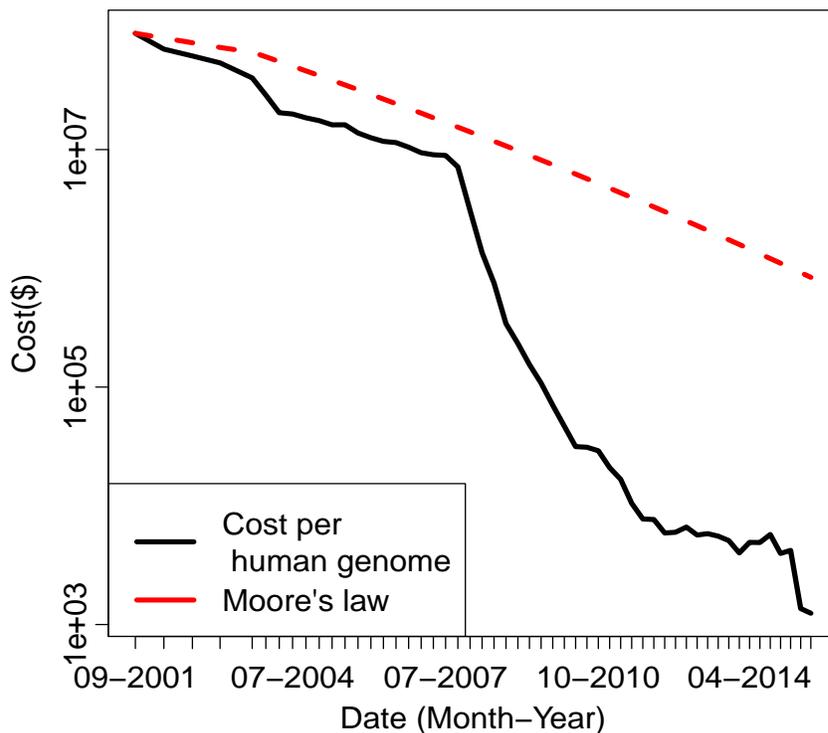


Figure 1.1: Cost per genome sequence (Source: NHGRI)

duced by these NGS platforms are more error-prone than those from conventional Sanger sequencing. Given this high volume of erroneous genomic data, genomic error correction and de novo genome assembly recently made their way to the forefront of big data challenges. The fundamental model of computation for these scientific applications is rapidly changing to address these challenges. At a broader level, these large-scale applications are driving the need for computing based on locality to scale to increasing amount of data. The thesis has developed software for different phases of sequencing analysis, such as genomic error correction and genome assembly leveraging the power of MapReduce, and NoSQL which became the de-facto standard of distributed computing.

The second part of the thesis addresses the hardware challenges for large-scale data. The growing size of the scientific dataset and their complex computation

demands more computing power from a processor as well as demands high I/O performance. The research has been done in collaboration with Samsung Research, S. Korea to access their state-of-the-art clusters powered by Samsung SSD and Intel Xeon processor to make a comparative analysis between these novel cluster architectures and the traditional HPC cluster. The thesis provided significant insight on how to deploy different hardware components (such as, processor, storage and memory module and network interconnect) in a cluster to optimize it for big data analysis. Instead of focusing only on processing speed (i.e., FLOPS) the thesis provided more stress on overall cluster architecture so that the cluster is properly balanced in terms of both performance and economy. Besides experimental evaluation, the thesis also develops a theoretical model extending Amdahl's second law to derive the configuration of an optimally balanced cluster architecture to resolve the performance and cost conundrum in big data analytic cyberinfrastructure. ^{2 3}

Finally, in the third part, the thesis addresses the challenges involved in sharing and transferring the big data among multiple clusters situated in a different geographical location. Sharing of big data across multiple clusters located in institutes to foster translational research has never become an easy job. The scenario is further complicated when the data needs to be protected by regulations. For example, a patient's healthcare information if shared, needs should be protected with HIPAA regulation. To this end, the thesis has developed a high throughput platform for sharing biomedical data leveraging the decentralized model of Blockchain technology while holding the technology's security and privacy promises. The thesis developed the framework using Blockchain with peer-to-peer storage technology to enable managing large-scale data lacking with the vanilla Blockchain.

²Second part of the thesis is done in collaboration with Samsung Ltd., S.Korea.

³Several developments/outcomes/observations of this thesis has been used for procuring and setting up the Delta cluster at LSU (CCT) in collaboration with IBM which has been published as a dynamic white paper [1]

1.3 Research Outline

The rest of the thesis is organized as follows. Chapter 2 discusses the background of the study including programming model of different big data analytics software used in this study and a brief overview of the SSD storage architecture. Chapter 3 proposes a large-scale genomic error correction tool that has been developed to address the challenges involved in improving the quality of second-generation sequencing data, specifically high-throughput Illumina short read data that introduces almost 1% of errors in the datasets which typically have billions of base pairs. Chapter 4 proposes a big data genome assembly software to address the challenges involved in the assembly of this high-throughput, short read data generated by Illumina. Chapter 5 then proposed another error correction tool for the third-generation PacBio sequencing platform which emerges recently with better promises of more complete assembly comparing to Illumina platform because of the significant rise in the read length but, severely limited in terms of quality with because of a significantly high error rate of 15%. Then, Chapter 6 focuses on the cyberinfrastructure that these software tools need for good performance. It evaluates different types of hardware infrastructure and provides an in-depth understanding of the system characteristics of these data- and compute-intensive applications. Based on this evaluations, Chapter 7 of the thesis proposes a theoretical model which points out the major shift from traditional HPC cluster with petaflops of processing power to a balanced cyberinfrastructure answering the question, how much I/O bandwidth or memory is required per GHz of processing power so that expected performance can be obtained in a cost-effective way. Chapter 8 covers the final aspect of the big data challenges i.e., big data transaction. This chapter proposes a high throughput architecture for sharing large-scale scientific data among different clusters or computing environment situated in geographically disparate locations. Finally, Chapter 9 concludes the thesis.

Chapter 2 Background

This chapter describes the background of big data challenges in the scientific computation. After providing a brief history of big data and big science the chapter focuses on the big data challenges specifically in bioinformatics and biomedical domain which is a major part of this thesis. Then, this chapter describes the basic programming models, hardware, and technologies which are used in this thesis to address the big data challenges to facilitate the discussion throughout the thesis. Some key terminologies are also clarified here which are used later in the thesis.

2.1 Brief History of Big Data and Big Science

The modern data revolution started after the term *big data* is coined by Roger Mougaldas of O'Reilly media, and Hadoop [2] is developed in 2005, one year after the term *Web 2.0* emerged and the *MapReduce* [3] model is published by Google. Not only in the field of big sciences, big data is found to be ubiquitous. Starting from genome analysis to astronomical analysis, from social media to e-commerce, both scientists and business-persons wish to find patterns buried in data either to solve fundamental questions of basic science or to gain competitive business advantages over others.

Henceforth, the research in the field of big data started accelerating. Tons of programming models, abstractions, and software frameworks have been proposed in the last few years to ease the big data analysis. These frameworks abstract away complex logic for distributed computing so that the application developers can focus only on the scientific or business logic only. For example, MapReduce [3], Bulk Synchronous Parallel [4], NoSQL, etc. are to name a few.

The research in the field of hardware also gained momentum by the dire need for performance and scalability in big data processing. Hardware vendors such as

Samsung, IBM, Intel and Nvidia revolutionized the field of storage and processor which accelerated the big data processing and significantly improved a throughput of the computation.

2.2 Big Data Challenges in Genomics

Genome analysis pipeline broadly consists of five different phases, high throughput DNA sequencing, Error removal and correction, genome assembly, alignment to reference genome and finally variant calling and other downstream application. As shown in Figure 2.1, among these five phases the error removal and correction and the assembly pose the maximum challenges in terms of big data processing.

In the first phase, high throughput DNA sequencing machines such as Illumina Genome Analyzer, HiSeq, etc. read the entire genome randomly into shorter fragments called short reads. The data normally conform to higher error rate compared to conventional Sanger sequencing. Consequently, the data is oversampled with the motivation that correct samples will appear more than the incorrect samples in the dataset giving an opportunity to improve the overall quality of the dataset by statistical analysis. The second step is to employ advanced statistical analysis to improve the quality of the data. Given the high throughput of the DNA sequencing machine, the error correction phase is one of the most data- and compute-intensive phase of the pipeline. The third phase is genome assembly where the small fragments of the entire genome (i.e., short reads) are assembled together to reconstruct the entire genome. This thesis focuses on de novo genome assembly, i.e., both the error correction as well as the reconstruction of the genome is done without any reference genome. For both error correction and genome assembly, the thesis has used k -mer-based methodologies for its higher accuracy guarantee comparing to other existing methodologies. In these computations, the short reads are again divided into smaller fragments of length k called k -mer. Despite of its higher accuracy guarantee k -mer based computation is complicated by the size of data. For

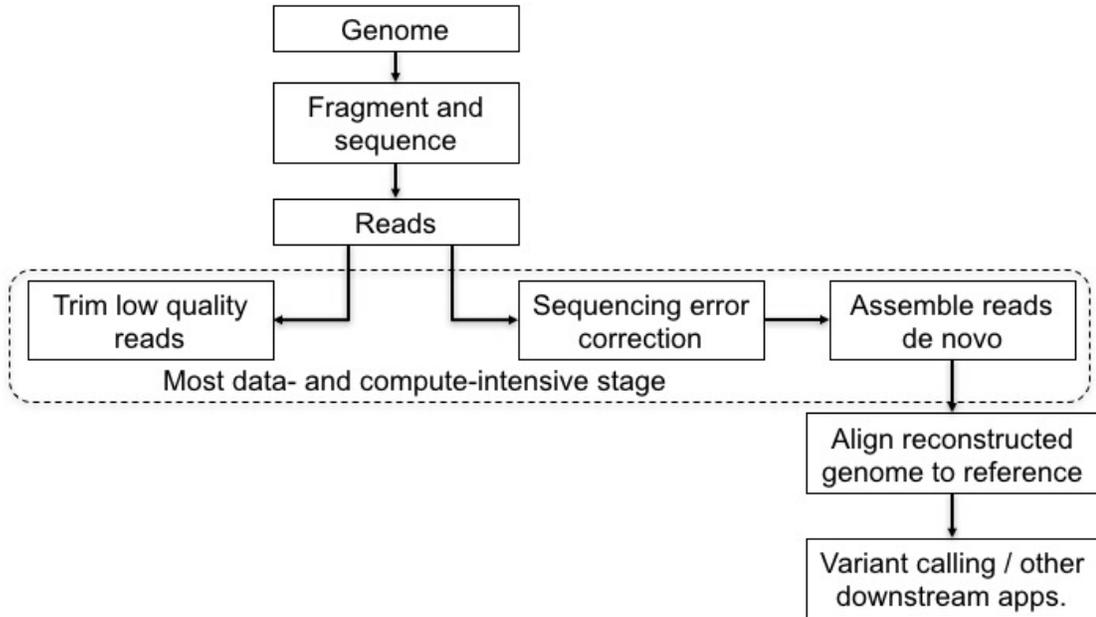


Figure 2.1: Genome sequencing pipeline

example, if k is chosen as 31 then the total possible number of unique k -mer string that needs to be processed is 4^{31} since each of the $k = 31$ locations of the string can be replaced with any of the 4 nucleotide characters A , T , G or C .

From the above discussion, it can be seen that the huge amounts of raw genome data further complicated with k -mer based methodologies pose several challenges in the field of bioinformatics in terms of optimal processing and storage of big data. This thesis is motivated to resolve these challenges using the state-of-the-art programming models for big data analysis (such as, MapReduce, Bulk Synchronous Parallel, and NoSQL) along with the recent advances in the hardware technologies.

2.3 Computation and Storage Models for Scientific Big Data

2.3.1 MapReduce

MapReduce [3] is one of the earliest programming models that revolutionized the field of big data analytics and became a de facto standard of distributed computing. The fundamental philosophy of this programming model can be found in the traditional functional programming languages, (such as Python, ML, Scala,

etc) where multiple data are processed using the same function in parallel using two programming constructs called 'map' and 'reduce'. The major motivation behind this programming model was to reduce the costly utilization network bandwidth and enable locality-based processing.

Figure 2.2 describes the MapReduce programming model. In this model, the data is distributed over different nodes of a compute cluster using a distributed file system (DFS). The model consists of three distinct phases, map, shuffle and reduce. The map phase considering locality reads the data from the DFS in the form of disjoint sets or splits called block. Then, a user-defined function is applied independently to each record of each block in parallel to get some information in a key-value pair format. These intermediate key-value pairs are then grouped together based on the corresponding keys, sorted and finally sent to the reducer. This phase is called shuffle. Finally, the reducer applies a function to aggregate or merge the information of each intermediate key-value pair and write the final output to the DFS.

There are many different implementation of MapReduce such as Hadoop [2] and Spark [5]. Spark computes data in memory for faster speed whereas Hadoop focuses on disk-based computation, thus enabling a huge amount of data processing at lower cost. This thesis focuses on using Hadoop to address the challenges the challenges in genomic analysis.

2.3.2 Bulk Synchronous Parallel (BSP)

Bulk Synchronous Parallel (BSP) [4] model was originally proposed by Valiant in 80's but found its utilization recently when large-scale graph processing became the mainstream for many of the data-driven scientific and business applications and Google published their BSP-based graph automated graph processing framework called Pregel [6]. The major motivation behind developing this programming model was to address the iterative challenges in MapReduce. Large-scale graph processing

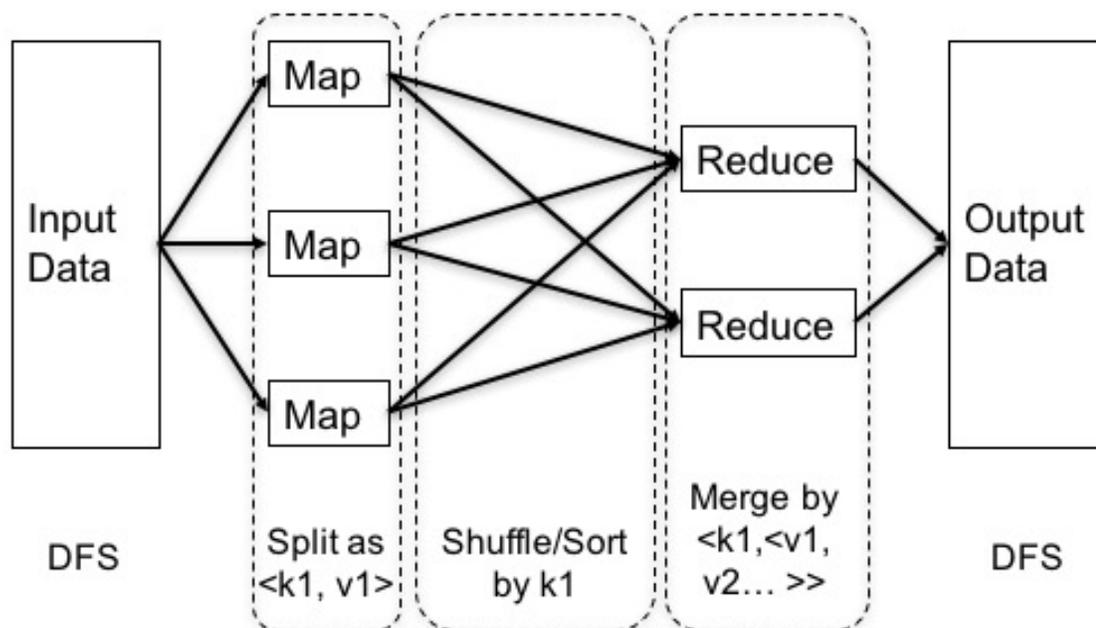


Figure 2.2: MapReduce model of computation

is core part of many different scientific or business applications. The computation consists of many iterations which are severely constrained by the I/O bandwidth of the disk in the vanilla MapReduce i.e., Hadoop.

BSP models enable vertex-centric graph processing described in Figure 2.3. A user-defined program called vertex-program is applied to each vertex of the graph similar to the map phase of MapReduce. Then this computation proceeds in the form of superstep. Each superstep consists of a map-like computation step discussed earlier followed by a synchronization step. This synchronization step is called barrier synchronization when each vertex sends the output of the corresponding instance of vertex program to all other vertices. The next computation step starts on the basis of the received messages. Like MapReduce, there are several implementation of BSP enabled graph processing such as Giraph [7] and GraphX [8]. Since the major focus is to enhance the performance of iterative computation, unlike Hadoop, all these implementations are operated in memory. This thesis uses Giraph to address the challenges the challenges in genomic analysis.

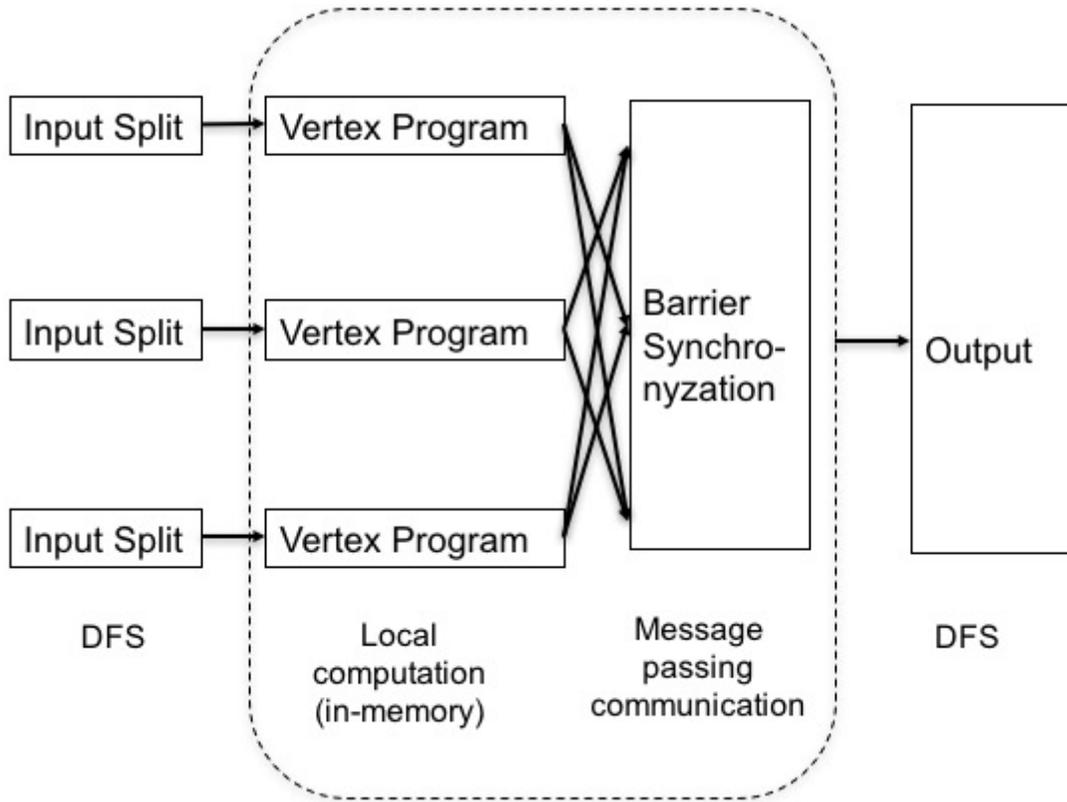


Figure 2.3: Bulk Synchronous Parallel (BSP) model of computation

2.3.3 Distributed NoSQL Storage

The fundamental model of NoSQL storage can be found in traditional hash table implementation where the data is stored in a tabular format as key-value pairs. The model became popular after Google publish their new distributed storage system called BigTable [9]. The major motivation of this storage model was to address the limitation in the relational database and enable faster data read/write.

Unlike the B-tree indexing structure in relational or SQL database, NoSQL or Not-only-SQL databases either use log-structured merge tree (LSM tree) or murmur hashing. The obvious benefit of using LSM tree or hashing is to lower space utilization for indexing. Also, it spends less amount of time for calculating indexes which is critical for faster data read or write. The entire dataset is partitioned over different nodes of a computing cluster. During writing the data, these NoSQL databases complies with CAP theorem while guaranteeing either even-

tual consistency or some form of lazy consistency. During reading the data, these databases schedule a number of instances of the query based on the number of partitions where each query-instance is responsible for searching a partition. Figure 2.4 shows the user’s view of a general purpose NoSQL database where many different clients (or, users) are querying different servers in parallel using APIs to gain uniform access to the data which is partitioned over the servers. There are

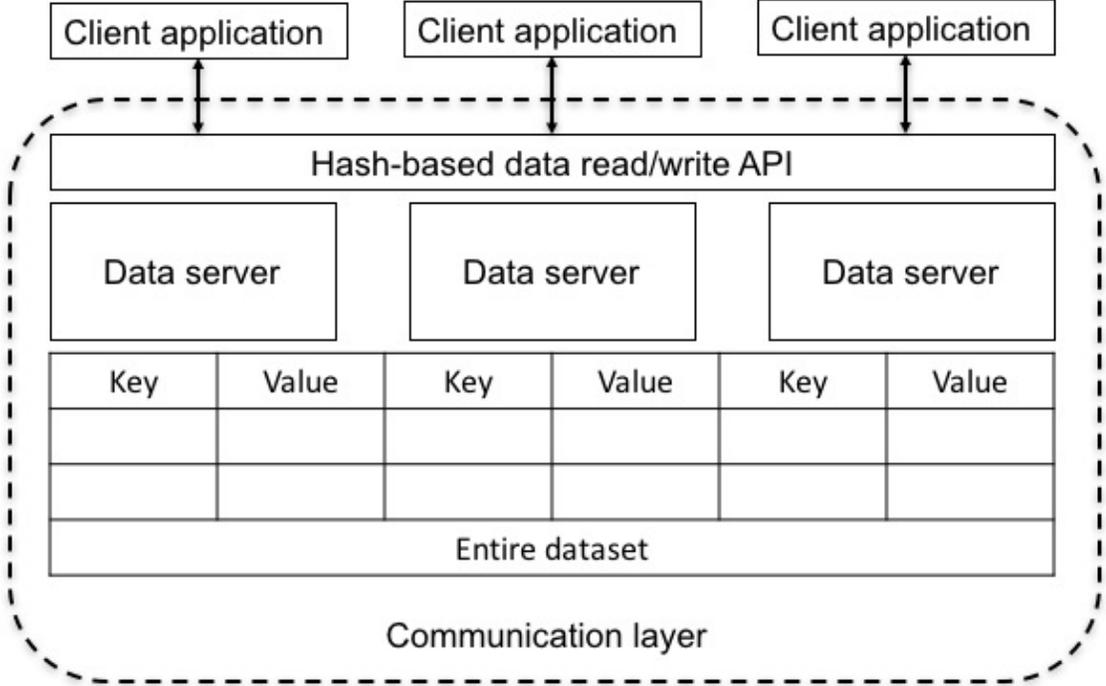


Figure 2.4: Distributed NoSQL storage model

many different implementations of NoSQL databases based on their consistency guarantees or based on their storage technologies. For example, DynamoDB [10], CouchDB [11], etc. guarantee availability and partition tolerance whereas Hazelcast [12], Redis [13], Hbase [14], MongoDB [15], etc. guarantees consistency and partition tolerance. On the other hand, MongoDB and HBase both are disk-based NoSQL primarily focusing on large-scale storage whereas Hazelcast and Redis are in-memory NoSQL focusing on faster computation. This thesis uses Hazelcast to enable faster computation in the area of bioinformatics.

2.4 Hardware Technologies for Big Data HPC

Although different big data analysis software tools significantly reduced the network bandwidth requirement, the locality-based computation demands significantly more compute cycles per processor than ever before, with extreme I/O and memory performance also required. Consequently, the landscape of HPC infrastructure is evolving rapidly. Hardware vendors (e.g., Samsung, IBM, Intel, etc) and large-scale HPC cluster providers (e.g., XSEDE, NSF Cloud, etc.) started spending millions of dollars and significant amount of effort to come up with optimal hardware and cluster architectures that is required for high performance of these big data analytic tools.

For example, Samsung developed high performance solid state drives (SSD) to improve the I/O bandwidth and reduce the I/O wait in big data applications. Comparing to 100-200MBPS of I/O bandwidth of a typical mechanical hard disk drive (HDD) an electronic SATA SSD provides almost 500-600MBPS showing an almost 4-5 times improvement. The technologies in SSD is also rapidly evolving. An NVMe SSD provides 2-3GBPS incurring another 4-5 fold improvement compared to the SATA SSD. In summary, from early 2000, the bandwidth increases at a sharp rate showing an improvement of almost 20 factors in the recent years. On the other hand, the cost of storage bandwidth started declining to change the performance point for big data analysis.

In the field of processing technology (e.g., CPU and GPU) also the improvement is quite significant. IBM developed Power8 processor which exploits 8 independent threads (simultaneous multithreading) of execution to maximize its resource utilization ability aiming at different big data applications. Intel released Knights Landing (KNL) processor with over 8 billion transistors aggregated up to 76 cores per die and 4 hardware threads per core. Knights Landing is marked as the largest chip Intel has ever made and is capable to deliver more than 6 Ter-

aFLOPS. On the other hand, NVIDIA's latest Tesla P100 GPU with 3840 CUDA cores (64 CUDA cores per SM) is capable to deliver a performance more than 21 TeraFLOPS if used properly in conjunction with NVLink interconnect.

In collaboration with different hardware vendors such as, Samsung and IBM, this thesis is motivated to evaluate cutting-edge hardware technologies and cluster architectures to accelerate big data analytics. Beyond experimental evaluation, the thesis is also motivated to develop an easy-to-use theoretical model to help and contribute towards the effort of system designers who are thriving for architectural balance in HPC infrastructure in terms of both performance and economy.

2.5 Data Sharing Model for Scientific Big Data

Blockchain has recently emerged as a new technology for the transactions over the Internet. Although, it is well explored in the domain of financial services (such as, bitcoin [16]) mainly because of its security promises and decentralized trust model, it is relatively new in the domain of data interoperability.

For security, Blockchain uses a Markle tree-based mechanism further strengthened by an SHA-2-based cryptographic hash. As shown in Figure 2.5, the Merkle tree in a block of a Blockchain network consists of nodes containing SHA2-based cryptographic hashes. The leaf nodes are hashes of a transaction or set of transactions. Nodes further up in the tree are the hashes of their respective children. There are several implementations of Blockchain. Bitcoin [16], Ethereum [17], HyperLedger [18], etc. are few of the open source Blockchain frameworks available. In this thesis, we explore the blockchain's opportunities in healthcare interoperability with Ethereum which features smart contract functionality to facilitate general purpose online contractual agreements.

Although Blockchain (such as, Ethereum) promises for security and privacy in transferring the data there are few limitations when it comes to transferring big data. To address the solution for managing large-scale data, common and crucial

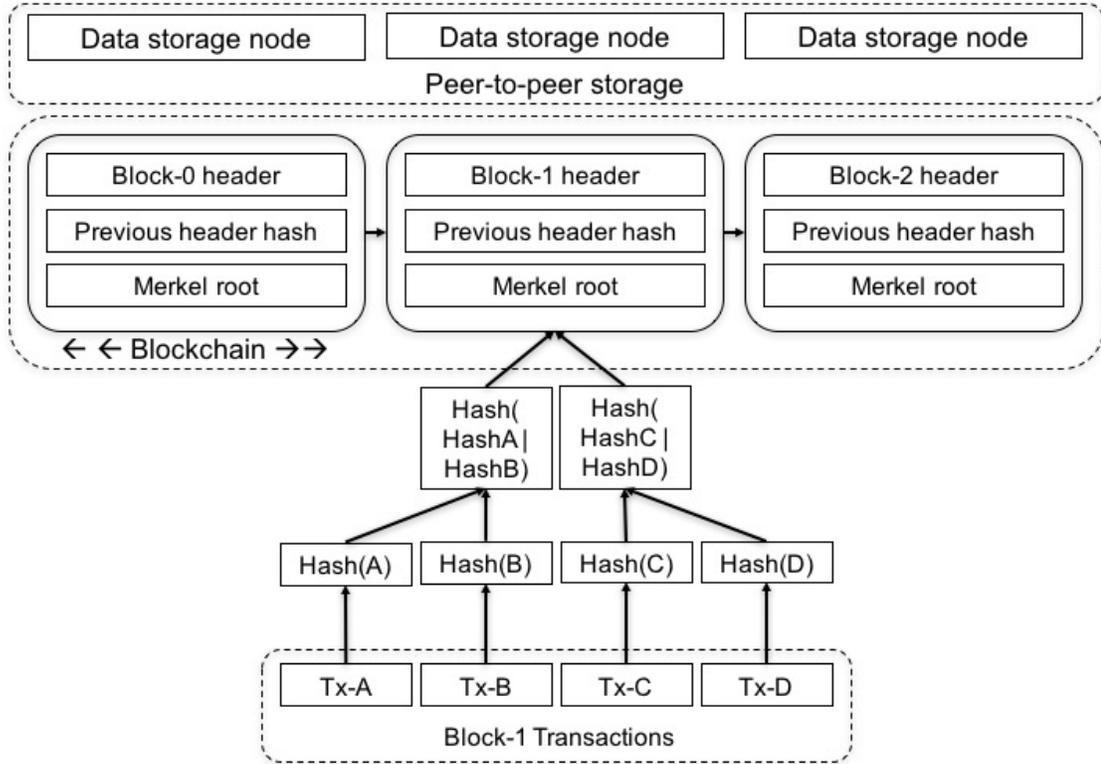


Figure 2.5: Blockchain transaction model

for medical records but lacking with the vanilla Blockchain technology, we explore the possibilities in peer-to-peer storage model.

As shown in Figure 2.5 this peer-to-peer storage is an extension and complementary to the existing Blockchain ecosystem. The same Merkle tree-based approach is used in the peer-to-peer storage model. Instead of a transaction (as in vanilla Blockchain), the leaf nodes consist of hashes to raw data blocks.

Among many peer-to-peer storage implementations (such as Swarm [19], Storj [20], etc. this thesis explores Swarm which is an open source peer-to-peer storage platform developed over Ethereum.

Chapter 3

Parallel Short-read Error Correction using Hadoop

A scalable and accurate error correction tool is essential for all next-generation sequencing (NGS) projects as high-throughput sequencing machines have started producing terabytes of data with significantly higher error-rates compared to conventional Sanger sequencing. This work develops ParSECH, a scalable and fully distributed error correction software based on k -mer spectrum analysis, without the need of a reference genome. To achieve high scalability over terabytes of data and hundreds of cores, ParSECH utilizes two open-source big data frameworks: Hadoop and Hazelcast. To achieve high accuracy, unlike existing error correction tools that use a single k -mer coverage cutoff to detect errors, ParSECH determines the skewness involved in the k -mer coverage of each individual read, followed by correcting the errors in each read separately for low and high coverage regions of the genome. The scalability of ParSECH is demonstrated by correcting the errors of both simulated and real whole human genome data with coverage ranging from 2x to 40x. ParSECH can correct the largest dataset (452GB human genome), which could not be handled by the existing error correction tools, in about 39 hours. For a small E.coli genome dataset, ParSECH demonstrates 94% accuracy, higher than 90% accuracy of Quake.

3.1 Introduction

Recent technological advances have dramatically improved next generation sequencing (NGS) throughput at a substantially lower cost. However, the reads produced by these NGS platforms are more error prone than those from conventional Sanger sequencing. For example, Illumina HiSeq produces errors at a rate of 1% per base sequenced [22]. For Pacific Bioscience, the error rate is approximately

This chapter previously appeared in the proceedings of BICOB 2017 [21]

15% [23]. Given that these high-throughput technologies typically produce billions of base calls per experiment, millions of errors are expected per experiment. Consequently, it is imperative to develop accurate and scalable error correction tools to improve the quality of these large-scale genome datasets. To be useful in practice, these software tools must be distributed, scalable and should make economical use of time and memory while also correcting the errors of the large datasets with high accuracy. However, most of the existing error correction software (e.g., [24, 25, 26, 27]) are limited by the computation power and memory space available in a single compute node, consequently, far from producing satisfactory result when applied to large NGS datasets.

To address this limitation, ParSECH is developed. It is the first distributed error correction tool based on k -mer spectrum, which can work on a cluster of commodity hardware. ParSECH utilizes two state-of-the-art big data frameworks: Hadoop for distributed data processing and Hazelcast for distributed in-memory storage to store k -mer spectrum. The underlying design principle of ParSECH carefully considers data locality to scale with terabytes of data.

The distributed error correction algorithm is a modified version of digital normalization [28], which can produce competitive accuracy for genomic dataset. To achieve high accuracy on different datasets, unlike existing error correction tools that use a single k -mer coverage cutoff to detect errors, ParSECH determines the skewness involved in the k -mer coverage of each individual read, followed by correcting the errors in each read separately for low and high coverage regions of the genome. In addition, ParSECH is fault tolerant, that is, it can continue to operate properly in the event of a failure of one or more nodes in the compute cluster which is a desired property of any big data analytic tool.

The scalability of ParSECH is demonstrated by correcting the errors of both simulated and real whole human genome data with coverage ranging from 2x

(22GB) to 40x (452GB). ParSECH can correct the largest dataset (452GB real human genome dataset available with NCBI accn. #SRX016231), which could not be handled by the existing error correction tools, in about 39 hours. For a small E.coli genome dataset (NCBI accn. #SRX000429), ParSECH demonstrates 94% accuracy, higher than 90% accuracy of Quake.

The rest of the chapter is organized as follows. Section 7.2 discusses the related work. Section 3.3 presents the error correction algorithm and architecture. Section 3.4 demonstrates the accuracy and scalability of ParSECH, and Section 5.5 concludes this chapter.

3.2 Related Work

The most popular methodology for error correction is the spectral alignment approach based on k -mer counting [29]. This approach first counts the multiplicity of each unique k -mer and then applies a threshold such that k -mers with multiplicity below the threshold are considered erroneous (i.e., erroneously altered during the base-call) and systematically edited into high-multiplicity k -mers.

Based on this basic framework, many error correction tools have been developed by considering the trade-off among accuracy, speed, and memory and storage efficiency. Reptile [26] and Hammer [30] use a Hamming distance-based approach between neighboring reads to achieve good accuracy. Quake [24] uses a quality score-based probabilistic k -mer count, followed by a majority voting for base correction to achieve high correction accuracy. ParSECH also uses similar k -mer counting like Quake. However, unlike Quake’s single coverage cutoff to classify true and erroneous k -mers, ParSECH finds the skewness in the k -mer abundance in a read to better differentiate between true and erroneous reads and then separately edits the errors of low and high coverage reads to achieve higher accuracy.

To improve the performance of error correction for large-scale datasets, existing error correction tools utilize advanced encoding schemes and in-memory

data structures, such as suffix tree and Bloom filter, to store k -mers. For example, Quake [24] uses Boost library’s `dynamic_bitset` to address the high memory challenge for correcting large-scale datasets. RACER [31] uses a 2-bit encoding of nucleotides and a 64-bit k -mer representation to achieve memory efficiency. SHREC [32] and HiTEC [33] use a suffix array index of the input reads to locate and correct errors while Lighter [34] and Musket [35] use a bloom filter for correcting errors. Despite of the improved memory efficiency of these existing tools, it is not clear how well they can address the data challenges involved in ever-growing volume of NGS data using a single node limited by memory as well as compute cores. To solve this issue, ParSECH provides a distributed solution for sequencing error correction tuned to work on a scale-out cluster where computation resources can be added according to the increasing data demand.

Several alternative attempts have also been made to handle the large volume of data using the recent advances in computing technologies. DecGPU [36] implements a distributed GPU-based algorithm that is much faster than most of the error correction tools. However, the proposed GPU-based algorithm is limited by the small size of GPU memory per compute node. Furthermore, the high cost of GPU and main memory cumulatively incurs huge computation cost for handling a large amount of data. On the contrary, [23] proposes a disk-based methodology using BOND trees [37] for error correction, to handle a large amount of NGS data at lower cost. However, the disk-based approach is slower than other tools.

ParSECH resolves this conundrum of performance, cost, and data-handling-capability prevalent in NGS data while also producing high accuracy. Unlike existing error correction algorithms discussed above, ParSECH is distributed, fault tolerant, and also tuned to work on a scale-out cluster of commodity hardware. Like [24, 26, 30, 33, 32], ParSECH also belongs to the family of k -mer spectrum-based methods that uses majority voting for error correction.

However, for improved accuracy, ParSECH considers low coverage and high coverage region separately instead of using a single k -mer coverage threshold. ParSECH’s Hadoop-based k -mer counting method is more scalable over large dataset than any of the existing methodology (e.g., Jellyfish-based method of Quake). ParSECH stores the large k -mer spectrum data into a distributed in-memory NoSQL database called hazelcast [12] enabling $O(1)$ -search, which is faster than both suffix array approach of [32, 33] as well as bloom filter approach of [36, 34]. Furthermore, the Hadoop- and Hazelcast-based distributed algorithm enables hundreds of searches in parallel, thus making voting process for error correction fast and scalable.

3.3 Methodology

3.3.1 Distributed Big Data Frameworks

1) Hadoop: Hadoop was originated as the open-source counterpart of Google’s MapReduce. It reads the input data from the underlying Hadoop Distributed File System (HDFS) in the form of disjoint sets or partitions of records. Then, in the MapReduce abstraction, a user-defined map function is applied to each disjoint set concurrently to extract information from each record in the form of intermediate key-value pairs. These key-value pairs are then grouped by the unique keys and shuffled to the reducers. Finally, a user-defined reduce function is applied to the value-set of each key, and the final output is written to the HDFS.

2) Hazelcast: Hazelcast [12] is an open source distributed in-memory NoSQL database (or, simply a key-value store). In a Hazelcast cluster, data is evenly distributed among the nodes using murmur hashing allowing horizontal scaling both in terms of available storage space and processing power. Hazelcast provides hash table functionality such as, *get* and *put* to insert and retrieve records in $O(1)$ time. Because of its operational similarity with hashtable, to refer to Hazelcast, the rest

of the chapter will use distributed NoSQL or distributed hashtable interchangeably. Hazelcast creates multiple in-memory instances of hashtable over multiple nodes and enables communication and load balancing among all these instances transparent to the users.

3) Reasons for using Hadoop and Hazelcast: Sequencing error correction is not only data- and compute- intensive but also search-intensive where the search space, that is, the size of the k -mer spectrum increases almost exponentially with the value of k (Maximum 4^k unique k -mers) and the total number of searches increases linearly with the sequence coverage. For example, a large genome with 1billion reads of length 100bp involves approximately 80billion searches in a set of almost 10billion unique k -mers followed by complex statistical computations. Existing technologies can hardly handle this challenge. Hence, an extremely scalable framework is required.

Fortunately, these data challenges can be addressed using the relatively new big data analytic software tools, Hadoop and Hazelcast. The $O(1)$ (asymptotically lowest) search complexity of Hazelcast makes the search process faster than any other data structure. Furthermore, being distributed in nature, Hazelcast can hold much larger size of k -mer spectrum data in memory compared to existing technologies. Finally, if designed properly, Hadoop- and Hazelcast-based distributed algorithm can enable hundreds of thousands of searches in parallel depending on availability of cores, thus makes the k -mer searching and the statistical computations for error correction fast and scalable.

3.3.2 ParSECH Architecture

Figure 3.1 shows the error correction pipeline of ParSECH. It has three different phases: 1) count k -mers, 2) locate errors, and 3) correct errors. The HDFS is used to store the raw short read sequences which acts as input to ParSECH. Whereas, Hazelcast is used to store the k -mer spectrum in memory in a distributed way.

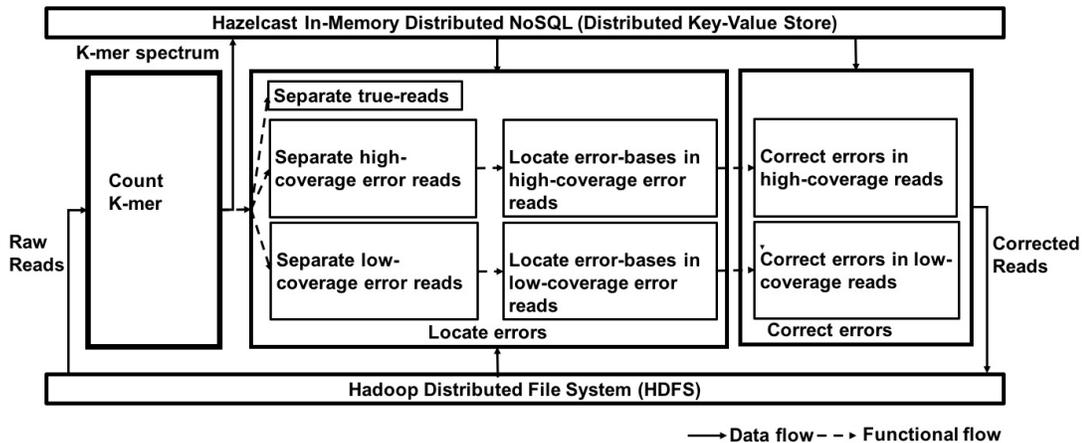


Figure 3.1: ParSECH architecture

Each phase of ParSECH consists of a parallel and distributed Hadoop MapReduce (or Map-only) jobs that do the required computation for that phase. In the first phase, only Hadoop is used to count the k -mers. In the subsequent phases both Hadoop and Hazelcast are used to locate and correct the errors. Finally, the corrected reads are written on the HDFS.

3.3.3 Counting k -mers

Algorithm 1 describes the MapReduce algorithm for counting k -mers. The map function scans each read of the dataset and emits each k -mer with an associated count of occurrences (the product of the probability of the correct call to each base of the k -mer as shown in line 3 to 6) as key-value pair. After the map function completes, the shuffle phase partitions these intermediate key-value pairs (i.e., k -mer and icount) on the basis of the keys. Finally, the reduce function sums together all counts emitted for a particular k -mer as shown in line 9 to 12. Once the final k -mer count (i.e., the k -mer spectrum) is generated, it is loaded to Hazelcast (in-memory NoSQL) as shown in line 14 using Hazelcast's *put* method.

3.3.4 Locating Errors

Algorithm 2 shows the ParSECH's parallel methodology for locating the error bases in shortreads. This is a Hadoop map-only job which is distributed over

Algorithm 1 Count k -mer

```
1: procedure MAP(String read) //compute parallel on all reads
2:   kmers[]=getAllKmers(read)
3:   for each kmer in kmers[] do
4:     intermediateCount=  $\prod_{i=0}^{k-1} p_c(b_i)$  // $p_c(b_i)$  is probability of correct call to base  $b_i$ 
5:     emitIntermediate(kmer, count) //Intermediate key-value pair for reduce
6:   end for
7: end procedure
8: procedure REDUCE(String kmer, iterable intermediateCount) //compute parallel on all unique  $k$ -mers
9:   for each c in intermediateCount do
10:    finalCount += c
11:   end for
12:   emitFinal(kmer, finalCount)
13:   putCountToNoSQL(kmer, finalCount) //write the  $k$ -mer count on Hazelcast in-memory
14: end procedure
```

multiple cores over multiple nodes. To filter the true and error reads ParSECH first calculates the Pearson's skew coefficient of the k -mer coverage of each read (Line 5 and 6). If the skewness lies in a certain interval (specified as $uBound$ and $lBound$ in line 7) the read is filtered out as true reads without any error (line 7-9). All the erroneous reads (i.e., reads with sufficient skew in their k -mer coverage) are then classified into high and low coverage area. As shown in line 11, if the median k -mer multiplicity of each read is greater than or equal to the median coverage of the entire k -mer spectrum then the read is classified to be in high coverage area of the genome. Otherwise, the read belongs to low coverage area. Then for high and low coverage reads, two different thresholds are chosen to identify the error k -mers (line 12 and 14). Then the subsection (line 22) of all the error k -mers is calculated to localize the error in order to find the erroneous base(s). Finally these error reads with all its error bases are written to HDFS with its coverage information (line 23).

3.3.5 Correcting Errors

Algorithm 3 shows the pseudo code of ParSECH's correcting error phase. ParSECH uses majority voting with two different criteria to correct errors in high and low coverage reads. As shown in line 2-7, for high coverage area, ParSECH edits the error base b so that the low multiplicity k -mers correspond to that base are transformed to high multiplicity (i.e., multiplicity greater than the $threshold_{high}$). The

Algorithm 2 Locate errors

```
1: procedure MAP(String read) //compute parallel on all reads
2:   for each kmeri in read do
3:     counts[] ← getCountFromNoSQL(kmeri)
4:   end for
5:   {mean, med, sd} ← findStats(counts[])
6:   skew ←  $\frac{\text{mean} - \text{median}}{\text{sd}}$  //Pearson's Coefficient
7:   if skew ∈ (lBound, uBound) then
8:     trueReads[] ← read
9:     emit(trueReads[]) //Filter out correct reads
10:  else
11:    if med ≥ medkmerspectrum then
12:      locateErrorBases(counts[], thresholdhigh, highCov)
13:    else
14:      locateErrorBases(counts[], thresholdlow, lowCov)
15:    end if
16:  end if
17: end procedure
18: function LOCATEERRORSINREAD(double[] counts, double threshold, String covInfo)
19:   for each kmeri with count < threshold do
20:     errorKmers[] ← kmeri
21:   end for
22:   errorBases ← getSubSection(errorKmers[])
23:   emit(read, errorBases[], covInfo)
24: end function
```

base yielding the maximum number of high multiplicity k -mers are selected as the correct base. If two bases produce same number of k -mers exceeding $threshold_{high}$, the base producing the maximum cumulative sum of multiplicity of k -mers are selected as the correct base.

On the other hand, for low coverage area, ParSECH edits each error base of a read in a way, so that the base that produces lowest amount of skew in the k -mer coverage of the read is selected as the correct base (line 8-13). The separate treatment of low and high coverage reads significantly increases the chance of an error being successfully corrected (True-Positive).

3.3.6 Advantages of Skew-based Filtering

The skew-based filtering methodology can better distinguish between low coverage true k -mers and high coverage error k -mers than the existing methodology (e.g., Quake) which uses only one k -mer coverage cutoff. For example, if a read belongs to the low coverage area of the genome then, all its k -mers (which are indeed true) may have a low multiplicity which can be frequently misclassified as errors based on the cutoff (or, threshold-cutoff) used. On the contrary, the skew-

Algorithm 3 Correct errors

```
1: procedure MAP(String read, int errorBases[], String covInfo) //compute parallel on all error reads
2:   if covInfo = "highCov" then
3:     for each kmer with error base b do
4:       newKmers[b][] ← replaceBase(b, b') //b' ∈ A, T, G, C
5:       C[b][] ← getCountFromNoSQL(newKmers)
6:     end for
7:     correctBase ← arg maxi {|C[i][j] > thresholdhigh | 1 ≤ j ≤ m|}
8:   else if covInfo = "lowCov" then
9:     for each kmer with error base b do
10:      newKmers[b][] ← replaceBase(b, b') //b' ∈ A, T, G, C
11:      C[b][] ← getCountFromNoSQL(newKmers)
12:    end for
13:    correctBase ← base producing k-mers with lowest skew
14:   end if
15: end procedure
```

based filtering uses a skew-interval to classify true and error reads before using the threshold-cutoff which significantly reduces the chance of misclassification.

Furthermore, the accuracy of the method has lower dependency on value of k than existing methodology as median statistics is used for distinguishing between true and error k -mer. Median statistics is robust, that is, for a small value of k a few substitution errors will not alter the median k -mer abundance of the read [28]. However, these errors will increase the skewness of the read. The robustness of the median statistics in the presence of sequencing error can be shown mathematically as follows: The total number of k -mers generated from a read of length l

$$n_k = l - k + 1 \quad (3.1)$$

After sorting all the k -mers of the read in ascending order of their abundance in the entire genome (obtained during k -mer counting), the position of the median can be expressed as,

$$p_{med} = \frac{n_k}{2} \quad (3.2)$$

The left side of p_{med} presents the k -mers with lower abundance. All the k -mers with errors will fall in this left side. Let us consider, for a maximum of e_{max} base

errors the median statistics will remain unaltered. Hence, it can be said,

$$k.e_{max} \leq p_{med} \tag{3.3}$$

where each error base may correspond to maximum k k -mers.

$$\implies e_{max} \leq \frac{p_{med}}{k} \tag{3.4}$$

It means, in case of a minimum of 2bp errors in an Illumina read of 100bp (which is common), k can be chosen any number between 15 and 19. However, it should not be very small. In average, good and almost invariant accuracy can be obtained for most of the genome sequences with k greater than 10.

3.3.7 Resource Utilization

ParSECH implements three different strategies to better utilize the available resources which may vary between a single desktop to a large cluster. First, a replication strategy where the entire k -mer spectrum is replicated to main memory of each node of the cluster and each hadoop worker use their local copy minimizing the network usage. If the number of unique k -mer is less, this strategy yields the best performance. Second, a distributed-memory-based strategy where the k -mer spectrum is distributed over main memory of the all the nodes of the cluster and the searches from each Hadoop worker is redirected to the node where a k -mer is hashed. It can handle relatively larger amount of data in memory but for network usage, it is not as fast as the replication strategy. Third, a disk-based strategy to handle any amount of data trading-off between memory (high-performance, high-cost) and disk (low-performance, low-cost) utilization. In fact, the Hadoop-based algorithms are developed with the minimum assumption of memory size which need to hold only a single read with all its k -mer abundance statistics.

3.3.8 Fault Tolerance

ParSECH is fault tolerant, that is, in case of one or more node failure in the cluster it continues to work properly. ParSECH's fault tolerance comes by replication and re-execution. Both the big data analytic software tools (i.e., Hadoop and Hazelcast) selected to develop ParSECH makes three replicas of the dataset in different nodes in the cluster. However, the replication factor can be configured based on the resource availability. In case of a node failure, ParSECH will start re-execution on a different replica of that subset of data.

3.4 Evaluation

3.4.1 Dataset

Table 3.1 shows the overview of all the dataset used for ParSECH's performance analysis. To evaluate ParSECH, both simulated (using `art_illumina` simulator) and real short reads have been used. Reads are simulated from two different whole genome sequences, 1) relatively smaller E.coli genome (4.6million bp) and 2) large size human genome (3billion bp). First, to shed light on different aspect of scalability and accuracy of ParSECH, this chapter used the simulated single end short read dataset (D1 and D2 in Table 3.1) of both the species ranging between 2x to 20x coverage. Then, to show ParSECH's performance to handle real dataset, and to compare it with other existing tool (e.g., Quake) the real paired end short read datasets for both the species are used. These real datasets are downloaded from NCBI. The size of the real human short read sequences (D4 in Table 3.1) is 452GB is the largest among all the dataset.

3.4.2 Accuracy on Simulated Data

We measure ParSECH's accuracy in terms of gain as introduced in [38]. Gain indicates the fraction of errors effectively corrected from the genome dataset. It

Table 3.1: Dataset (simulated and real) used to evaluate ParSECH

ID	Species	Source	Reference genome	Read length	Genome length	Coverage	Size (GB)
D1	<i>E. coli</i>	Simulated with 0.5% error rate	NC_000913	100	4639675	2x, 5x, 10x, 20x	0.87, 0.166, 0.314, 0.632
D2	Human	Simulated with 0.5% error rate	UCSC	100	3×10^9	2x, 5x, 10x, 20x	13, 27, 58, 116
D3	<i>E. coli</i>	SRX000429	NC_000913	2×36	4639675	162x	3.2
D4	Human	SRX016231	UCSC	2×100	3×10^9	47x	452.0

can be formally defined as follows,

$$Gain = \frac{TP - FP}{TP + FN} \quad (3.5)$$

where, TP (true-positive) indicates the number of errors which are successfully corrected, FP (false-positive) is the number of true bases which are changed wrongly, and FN (false-negative) is the number of errors which are falsely detected as correct. Table 3.2 shows the accuracy of simulated e.coli and human genome. For Table 3.2: Accuracy of *E.coli* and Human genome for different coverage with $k = 15$. (using BWA alignment)

Dataset	Cov.	TP	FP	FN	Gain (%)
E.coli	2.5x	90058	3427	154	96.03
E.coli	5x	224915	8966	218	95.92
E.coli	10x	448994	18372	833	95.73
E.coli	20x	900776	34544	1079	96.05
Human	2.5x	60909869	1740388	4461108	90.51
Human	5x	121819797	3499257	8906416	90.51
Human	10x	243642050	7004946	17810303	90.50
Human	20x	488367549	12825818	35716293	90.73

E.coli dataset, ParSECH achieves almost 96% gain. For Quake, the accuracy of such dataset is reported as 90% as reported in the their paper [24]. On the other

hand, for the large and complex human genome ParSECH's accuracy is more than 90% which cannot be handled with most of the existing tools in real time.

3.4.3 Scalability on Simulated Data

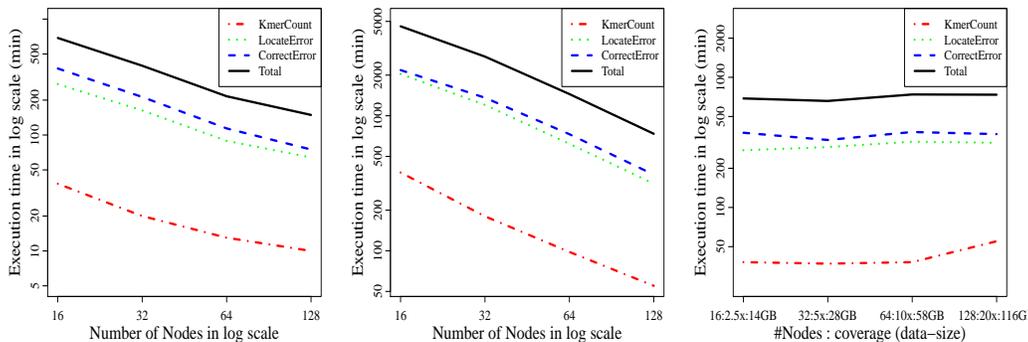
To show the scalability of ParSECH the simulated human genome datasets with different coverage are used. For this purpose, LSU HPC cluster called SuperMikeII is used where each node has 16cores and 32GB of memory. However, for ParSECH only 4cores per node are used to demonstrate ParSECH's performance on scaled out cluster. Among those 4cores, 2 are used for Hadoop and other 2 are used for Hazelcast. All the nodes of the Hadoop cluster is connected through 1Gbps ethernet network. A maximum of 128 nodes is used with this configuration. This configuration is a good representative of a commodity cluster of cheap hardware on which ParSECH shows almost linear scalability with increasing number of nodes.

1) Strong scalability: Figure 3.2a and 3.2b show the strong scalable nature of execution different phases of ParSECH. That is, for a fixed size of data, ParSECH's execution time decreases almost linearly with the increasing number of nodes. Here, the strong scalability of ParSECH is shown for 2x and 20x coverage (i.e., the two extreme cases for simulated reads from whole human genome) dataset. However, it is true for all the other dataset also.

2) Weak scalability: Figure 3.2c demonstrates the excellent weak scalability nature of ParSECH. That is, the execution time of ParSECH remains almost same if the number of nodes are changed in the same ratio as that of the data size.

3.4.4 Accuracy on Real *E. coli* Genome

To compare the accuracy of ParSECH's error correction algorithm with that of Quake an *E. coli* dataset is used. This dataset consists of 20816448 million paired-end reads each of length 36 bp with an error rate of 0.51%. This benchmark dataset has been used by many error correction tools to evaluate their performance.



(a) Strong scalability of ParSECH: Execution time for simulated short reads generated with 2.5x coverage of age (14GB) of whole human genome
(b) Strong scalability of ParSECH: Execution time for simulated short reads generated with 20x coverage of age (116GB) of whole human genome
(c) Weak scalability of ParSECH: Execution time for simulated short reads with varying coverage of whole human genome on varying #nodes

Figure 3.2: Scalability of ParSECH with different data size and different #nodes

Table 3.3: Performance on real *E. coli* genome ($k = 15$)

Metric	ParSECH	Quake
Gain(%)	94.10	90.03
% of original reads aligned (using BWA) successfully after correction	93.5	86.55
#cores	16	16
Time (hour)	1.73	1.3

ParSECH successfully corrected (i.e., TP) 956563632 base errors in this dataset. The corresponding FP and FN are 14450760 and 44570376 respectively. Consequently ParSECH achieves 94.10% gain whereas Quake achieves 90% (shown in Table 3.3). Almost 93.50% of the input reads are successfully aligned after correction with ParSECH. The corresponding count for Quake is only 86.55%. This is because Quake removes many reads from the dataset as well as trim several reads into shorter length which are not considered by the alignment software. On the other hand, ParSECH attempts to correct all the error bases in each read instead of removing it.

Table 3.4: Performance on real human genome ($k = 15$)

Metric	ParSECH	Quake
Gain(%)	94.3	-
% of original reads aligned (using BWA) successfully after correction	96.2	-
#cores	512 (128 nodes)	-
Time (hour)	39.09	failed

3.4.5 Scalability on Real Human Genome

To demonstrate the capability of ParSECH to handle very large scale complex genomes, ParSECH processed Yoruban male (NCBI accn. #SRA000271) dataset. This Illumina paired end dataset has a total of 1424378028 reads each of length 101, i.e., coverage of 47X. As shown in Table 3.4, the process took almost 39 hours over 128 nodes (i.e., 512 cores). After correction, ParSECH achieves 94% gain (TP=780598920, FP=12936036 and FN=17309923). 96% of all the input reads are successfully aligned to the reference genome after correction. On the other hand, Quake could not process this huge dataset in the available infrastructure mainly because it cannot scale over multiple node. Using the processing power of a single node it could not process the entire dataset in the maximum amount of time allocated for a single job in the HPC cluster. It is worthy to mention here that for Quake a single scaled up node with 1TB memory is used so that Quake never runs short of memory. However, for ParSECH the same scaled out setup is used as mentioned earlier (i.e., 32GB memory per node).

Figure 3.3 compares the best performance of ParSECH and Quake in terms of execution time over many subsets of human genome of varying size. Again, Quake used the same scaled up node (1TB DRAM) and ParSECH used the same scaled out cluster (32GB DRAM/node). For the small data Quake outperforms ParSECH because of ParSECH’s parallel computing overhead. However, for large data (> 10GB) ParSECH shows better result. Finally, Quake could not process the 452GB data in maximum allocated time but ParSECH completed in 39 hours.

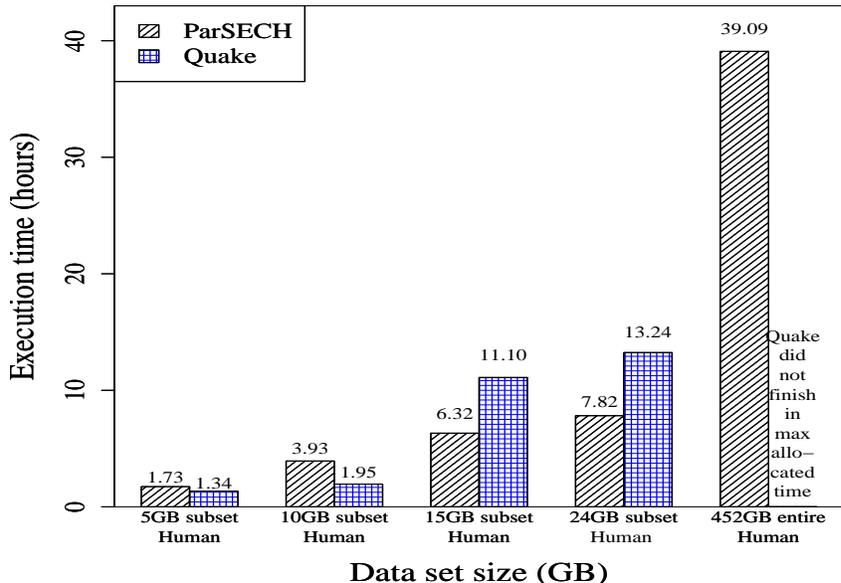


Figure 3.3: ParSECH’s scaled out (4cores and 32GB memory/node) performance vs Quake’s scaled up (20cores and 1TB memory) performance for large data

3.5 Conclusion

This work presents ParSECH, a scalable, fully distributed, fault tolerant sequencing error correction software which utilizes the power of current state of the art big data analytic software tools, Hadoop and Hazelcast to scale with huge volume of sequencing data producing good accuracy.

ParSECH’s algorithm has potential to detect and correct errors in RNA and metagenome sequences also as it detects the errors based on k -mer coverage skew of each read (unlike single coverage cutoff used in other tools) and corrects errors separately for low and high coverage areas. Also, the big data processing framework (based on Hadoop and Hazelcast) developed in this work can also handle the data challenges in other applications in the sequencing pipeline e.g., genome assembly, variant calling etc.

Chapter 4

Giraph-based Genome Assembler for Large-Scale Genomes

Managing prodigious volumes of NGS input data in a cost-effective way forces a growing number of sequence analytic applications to run on scaled out cluster of low-cost commodity hardware. Large-scale de novo genome assembly is no exception. However, traditional MPI-based assembly software cannot scale well with huge volume of data unless sophisticated and costly compute resources are provided which are unavailable to most researchers. The model of underlying computation should be changed significantly to address this critical need. In this work, GiGA, a parallel Giraph-based Genome Assembler is developed which uses de Bruijn graph approach for the assembly. GiGA uses recent big data analytic software, Hadoop, and Giraph which carefully consider data locality, thus automatically scale with terabytes of data on low-cost commodity clusters. The benchmark-evaluation over GAGE datasets shows that GiGA achieved significantly higher scalability, substantially lower misassembly and competitive NG50 compared to other assembly software. GiGA performs almost 1.5x faster than Contrail, a Hadoop-based genome assembler developed for commodity cluster. GiGA's capability to assemble large-scale vertebrate genomes over hundreds of cores is shown by assembling a human genome dataset (SRA000271) of size 452 gigabyte and almost 2 billion reads with 512 cores in almost 8.5 hours.

4.1 Introduction

The genome input data size already exceeded terabytes and will continue to grow as the technology improves. At the same time, the hardware cost for storage and processing this huge amounts of big data is increasing linearly that will soon turn the computational cost to the most dominant one. To address this challenge,

This chapter is published in JBCB [39]. Reprinted by permission

many of the sequence analytic applications are being forced to run on scaled out clusters of commodity hardware so that low cost hardware resources can be added to these clusters as the need arises in future.

As a consequence, the fundamental model of computation is changing rapidly in many of the sequence analysis applications. Deviating from traditional compute-intensive parallel programming model (e.g., MPI, etc.), scientists are increasingly using the data-centric frameworks, such as Hadoop, Giraph [40], Pig [41] etc. which have recently emerged as big data analytics software and are tuned to work on large clusters of commodity hardware. For example, Crossbow [42] uses Hadoop to detect single nucleotide polymorphisms (SNPs) in whole-genome sequencing data. The BioPig toolkit [43] uses Pig (an analytics tool based on Hadoop) for k -mer counting, pathogen detection, etc.

De novo genome assembly is an important application in the sequence analysis pipeline. However, limited works (e.g., Contrail [44]) have been done to optimize large-scale assembly application atop scaled out clusters of commodity hardware. Most of the existing assembly software such as ABySS [45], etc., use MPI, and can not scale with terabytes of data unless the sophisticated hardware is used. On the other hand, the performance of Contrail [44], the Hadoop-based assembler is severely constrained by huge amounts of disk I/O.

This work resolves both the problems by proposing GiGA, A Giraph-based Genome Assembler. GiGA uses the power of Hadoop and Giraph (a large scale graph processing framework developed atop Hadoop) to achieve high performance and scalability over hundreds of compute nodes in a commodity cluster. The use of Hadoop automatically scales the application over terabytes of data by moving the computation to the data as opposed to moving the data to the computation as is done in traditional parallel programming paradigms (e.g., MPI). Furthermore, the use of Giraph [40] enables in-memory de Bruijn graph processing which performs

a magnitude faster than the disk-based approach adopted in Contrail.

The rest of the chapter is organized as follows. In Section-4.2 discusses the related work to the study, issues in existing assemblers and describes the motivation. Section-4.3 describes the Hadoop and Giraph-based approach adopted in GiGA in details. Section-4.4 shows the performance of the assemblers both in terms of accuracy and scalability. Finally, Section4.5 concludes the study.

4.2 Related Work

A plethora of genome assembly software tools have been developed in the last few years. This section discusses the current state of de Bruijn graph oriented assembly software.

Velvet [46], Minia [47], Allpaths [48], Platanus [49], etc. are some of the examples which use efficient in-memory data structure to assemble large genomes in a single machine. But given the exponential growth of sequencing data, it is not very clear how will these stand-alone assembly-performs behave in future for large and complex genome datasets. A much better approach is to distribute the computation for large genome assembly into multiple nodes of a compute cluster. Parallel genome assembly software traditionally uses MPI to distribute the assembly workload across several machines. For example, ABySS [45], PASHA [50], RAY [51], Meraculous [52] use MPI for the assembly process. The scalability of these assemblers over large-scale sequencing data is severely constrained by three factors that are addressed in this work. First, the data input phase is not parallel in most of these assemblers. That is, the input sequence data is read by only one process (possibly with multiple threads but in a single machine) and is distributed to other processes over multiple machines for assembly. It means a significant amount is spent for I/O which does not scale at all. Second, huge volumes of data are transferred over the network during the assembly process which poses a huge communication bottleneck. This communication bottleneck impacts the

performance adversely when terabytes of input sequence data are assembled over hundreds of cores. Third, the sequential part of the graph simplification stage (in particular, compressing each linear path in the graph into one vertex) limits the scalability severely. Although these parallel assemblers process several linear paths in parallel, k -mers sharing the same path are merged one by one. That is, these algorithms are bounded by $O(n)$ where n is the number of nodes in the longest linear path in the de Bruijn graph

Contrail [44] was introduced to address these scalability issues associated with the de novo assembly of larger genomes on top of commodity hardware. It uses Hadoop [53] (an open source implementation of Google’s MapReduce) to make the computation parallel across a cluster consisting of hundreds of machines. Hadoop stores the input data in a distributed file system called Hadoop Distributed File System (HDFS) and reads the data in parallel with multiple processes over multiple machines. Thus, the I/O bottleneck of the MPI-based assemblers were eliminated. Hadoop also considers data locality. That is, it moves the computation to the place where data reside instead of moving huge volumes of data over the network. This eliminates the network bottleneck prevalent in the MPI assemblers. Thus, Contrail achieved high scalability using the power of Hadoop. However, its performance is severely constrained by Hadoop’s limitation on iterative computation. De novo genome assembly using de Bruijn graph is essentially a large-scale graph processing problem involving many iterative computations. These iterative steps in Contrail are represented as separate MapReduce jobs which leads to huge amounts of disk I/O. Huge amounts of graph data are read/written to the disk during setup and teardown of each MapReduce job. Furthermore, after each map phase, a huge volume of shuffled data is written to the disk which again makes the process extremely slow. If the disk I/O bottleneck can be reduced then the assembly process can gain significant performance boost with high scalability.

Motivated by this, in the Giraph-based assembler, we use distributed memory approach to design the de Bruijn graph simplification algorithms (compressing linear paths and removing tips and bubbles). However, developing in-memory graph simplification algorithms considering data locality for terabytes of sequencing data is not an easy job. This research uses Valiant’s Bulk Synchronous Parallel (BSP) [54] processing model as a theoretical background and developed the algorithms with Giraph, an in-memory graph processing framework developed on Hadoop. Furthermore, unlike existing assemblers, the graph simplification algorithms are fully parallel and bounded by $O(\log n)$ where n is the length of the longest linear path. Section-4.3 provides a detailed description of the algorithms.

4.3 Methodology

4.3.1 Programming Model of Hadoop and Giraph

Before explaining the algorithms in detail let us discuss the programming model of Hadoop and Giraph to facilitate the discussion.

Hadoop is an open-source implementation of Google’s MapReduce [3]. Hadoop reads the data from Hadoop Distributed File System (HDFS). The data is read in the form of disjoint sets records. Then, in the MapReduce abstraction model, a map function is defined by the user which is applied to each disjoint set simultaneously to process each record of each set and output an intermediate key-value pairs for each of the records. These intermediate key-value pairs are first written to the local file system, sorted and then, hashed to the corresponding reducer based on the unique keys. Finally, a user-defined reduce function (that performs some aggregation operation) is applied to the value-set of each key, to produce the final output which is again written to the HDFS.

Giraph was originated as the open-source counterpart of Google’s Pregel [6]. In the first phase Giraph leverage Hadoop mappers and reads the data for HDFS in

parallel by several processes over multiple machines. The computation of Giraph is inspired by Valiant’s Bulk Synchronous Parallel model. This is basically an iterative computation model where each iteration is called a superstep. In each superstep, a user-defined program, called a vertex-program, is executed over all the vertices. At the end of each superstep, any vertex can send a user-defined message to any other vertices to initiate the next superstep. Alternatively, the vertices can vote to halt if a certain condition is fulfilled. The computation stops when all the vertices unanimously vote to halt in the same superstep.

4.3.2 De Bruijn Graph Construction

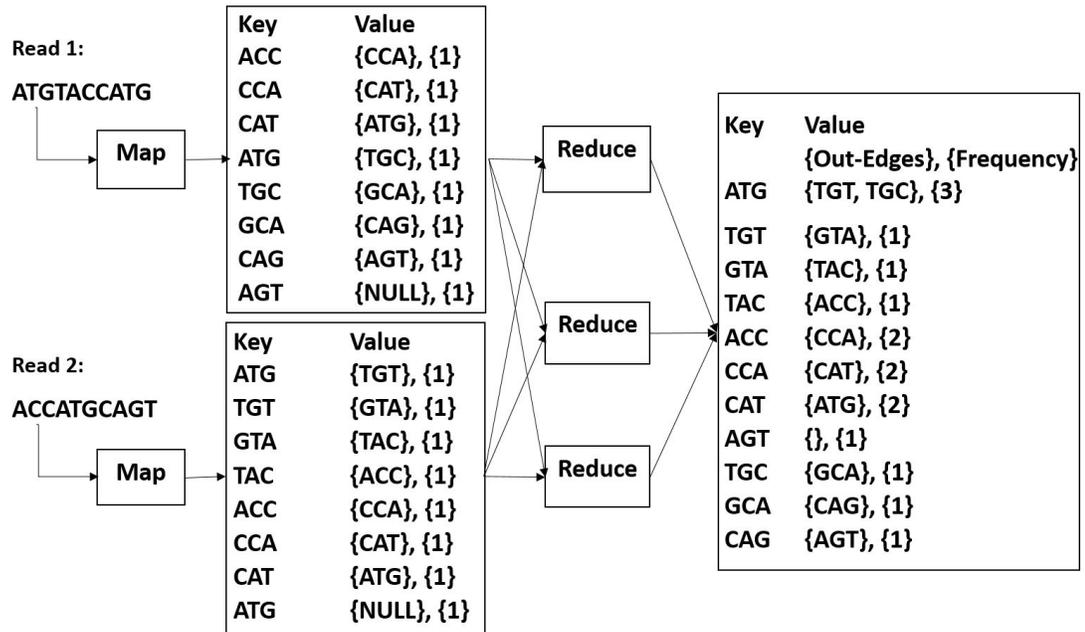


Figure 4.1: De Bruijn graph construction using Hadoop

Building a de Bruijn graph with Hadoop is shown in Figure 4.1. In the map phase of the Hadoop-based algorithm, each read is divided into several substrings of length k . These substrings are known as a k -mer. Two consecutive k -mers conforms a key-value pair, representing a vertex and an edge from that vertex respectively.

Additionally, the value field also contains an integer (1) which corresponds to the frequency count of the key. A similar process is repeated for the reverse complement of the reads. After the successful completion of the map function, the shuffle phase partitions the intermediate key-value pairs on the basis of the keys to collect the edges of the graph emitted from the same source k -mer. Finally, the reduce function aggregates the edges of each source k -mer and saves the graph structure in the HDFS. At the same time, the frequency-counts are also added up, yielding the actual frequency of the source k -mer. Figure 5.4 shows an example construction of a de Bruijn graph from two short reads.

4.3.3 Graph Simplification

GiGA's graph simplification algorithms are developed using Giraph where computation proceeds in supersteps. To develop the algorithms, all the computations involved in graph simplification process are divided into several rounds where each round consists of two supersteps. Broadly, the first superstep identifies the interrelated vertices based upon certain conditions, whereas the second superstep computes over the interrelated vertices to simplify the graph.

4.3.4 Compress Linear Chains

The first step that follows after building the graph is compressing the linear chains of vertices in the graph. (Refer to Figure 4.2 and Algorithm 4). The non-branching paths of vertices can be compressed into a single vertex. The algorithm is based on parallel random list ranking [55] where all the compressible vertices in the linear paths are tagged with either *head* or *tail* with equal probability, and finally, all the *head-tail* links are merged. The process repeats until all the vertices in a chain are merged or a predefined number of superstep is reached.

Algorithm-4 shows the computation involved in one round of Giga's compression algorithm. As mentioned before, each round of compression corresponds to

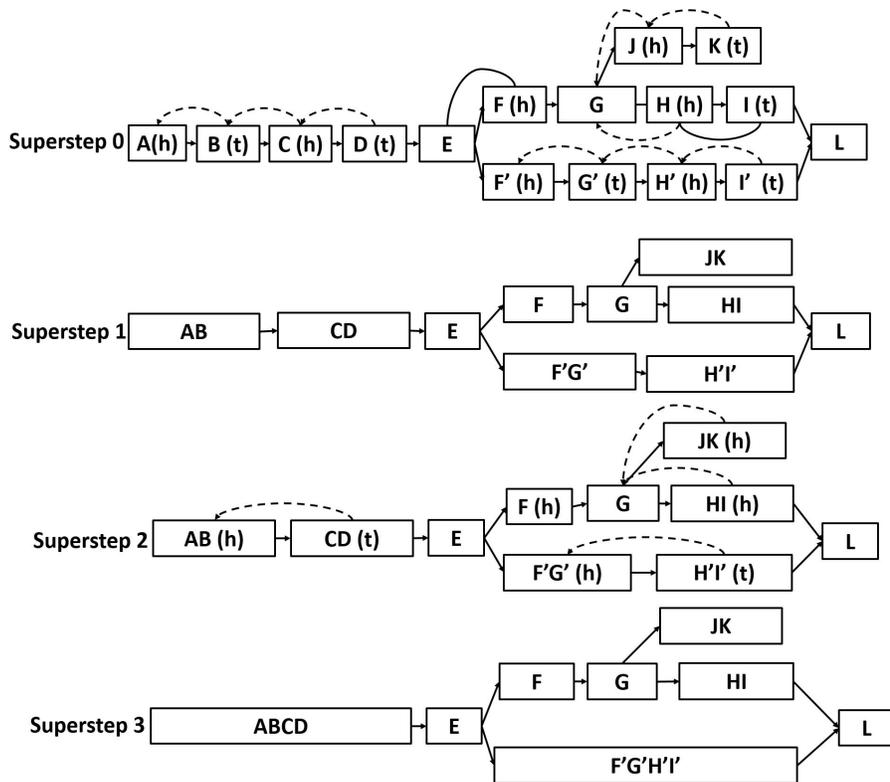


Figure 4.2: Initial Compression: Each two supersteps make a round. Dotted lines show the messages

two Giraph supersteps. In the first superstep, all the vertices with exactly one outgoing edge are identified as compressible and tagged with either *head* or *tail* with equal probability. At the end of the same superstep, the required information from each compressible vertex is sent to its predecessor. The next superstep corresponds to the computation in predecessors (vertices that received the messages in the last superstep). In this superstep, the *head* vertices are merged with *tail* vertices, and the edge information is updated. Figure 4.2 shows an example of how the compression algorithm works. Superstep 0: In the first superstep, all vertices with exactly one outgoing edge are identified as compressible and tagged with *h* or *t* randomly and send a message containing the corresponding *k*-mer, frequency, successor's id and the random-tag to their predecessor. For example, *A* and *B* is tagged randomly with *h* and *t* respectively. Observe, vertex *E* and *G* are neither

Algorithm 4 Compress Linear Chains

```
1: procedure COMPUTEPARALLELFORALLVERTICES(superstepID, messages)
2:   if (superstepID is even) then
3:     if (vertex has a single outgoing edge) then
4:       randomly assign head or tail to vertex.tag;
5:       SENDMESSAGE(predecessor, {vertex.id, vertex.tag, vertex.contig, vertex.frequency,
   vertex.neighbors});
6:     end if
7:   else
8:     if (only one message is received) then
9:       successor.id, successor.tag, successor.contig, successor.frequency ← PARSEMESSAGE(message);
10:    end if
11:    if (vertex.tag is head and successor.tag is tail) then
12:      concatenate last character of successor.contig to vertex.contig;
13:      vertex.frequency = vertex.frequency + successor.frequency
14:      DELETEVERTEX(successor.id);
15:      DELETEEDGE(vertex.id, successor.id);
16:      ADDEDGE(vertex.id, successor.neighbors);
17:    end if
18:  end if
19: end procedure
```

tagged nor send any message since they have two outgoing edges. In superstep 0, all the vertices are marked with either h or t and all the t vertices send a message to its predecessor. In superstep 1 (i.e., the second superstep), The vertices which received the message from their successors check for h - t link and they are merged. For example, A was tagged h . It received a message from B containing t . So, A appends B and its edge is adjusted. In superstep 2, 3 The same steps are repeated in the second round

In each round of compression, half of the total vertices are expected to be compressed. The number of rounds is bounded by the longest linear path of the graph. If the longest path has p vertices then the total number of rounds can be asymptotically represented as $O(\log(p))$.

4.3.5 Tip Removal

The tip removal is straightforward since dropping edges connecting to tips does not affect the other parts of the graph (Refer to Algorithm-5 and Figure 4.3). GiGA followed the general approach adopted in other assemblers to avoid the loss of any genuine sequence. That is, GiGA considers a sequence of vertices as a tip only if their length is less than $2k + 1$.

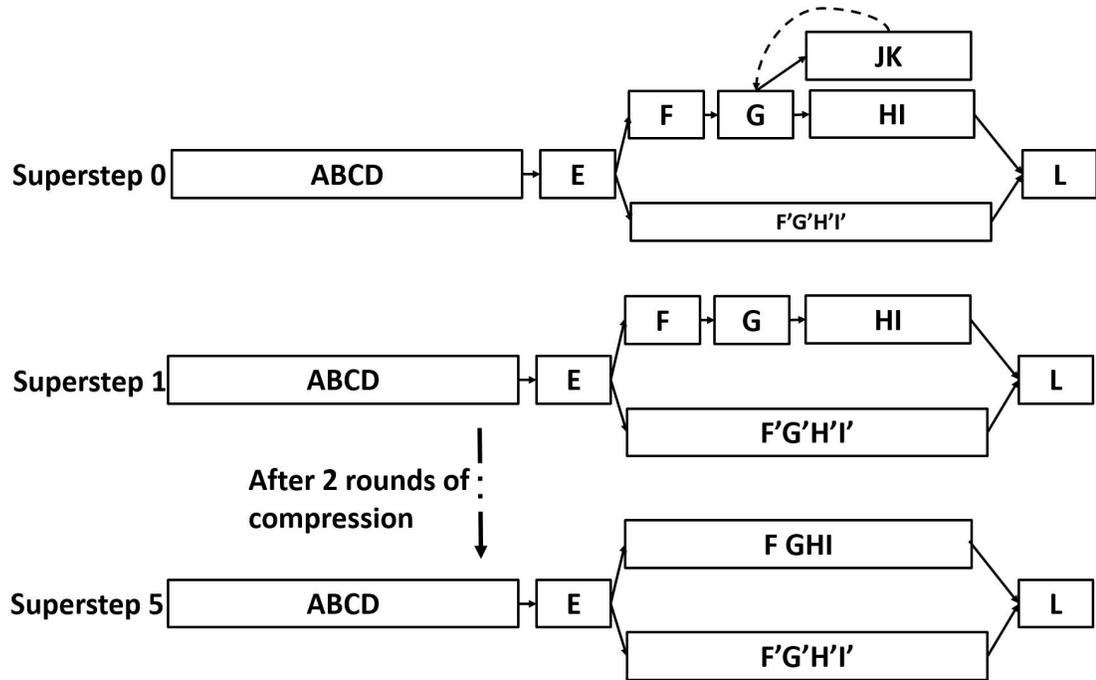


Figure 4.3: Tip removal: Each two supersteps make a round.. Dotted lines show the messages

Algorithm-5 shows the steps involved in one round of the Giraph-based tip-removal process. In the first superstep, the vertices having an in-degree of one and length less than twice of the k -mer length are identified as potential tips, and a message is sent to their immediate predecessor. In the next superstep, the vertices which received the message(s) from their corresponding successor(s) delete the edge(s) that connects to that successor(s), thus removing all the existing tips from the graph in just one single round. Removal of the tips will generate some new linear paths in the graph that we compress again. Figure 4.3 illustrates the entire process with a simple example. In superstep 0, vertex JK is identified as a potential tip since its length equals $2K$ and it has only one incoming edge and no outgoing edge. So, it sends a message to its predecessor G containing its own id. In superstep 1, vertex G removes JK from its neighbors' list. Superstep 5

Algorithm 5 Tip Removal

```
1: procedure COMPUTEPARALLELFORALLVERTICES(superstepID, messages)
2:   if (superstepID is even) then
3:     if (Length of vertex.contig is less than or equal to  $2k$  and the vertex has no outgoing edge) then
4:       SENDMESSAGE(predecessor, vertex.id);
5:     end if
6:   else
7:     for (each message in messages) do
8:       successor.id  $\leftarrow$  PARSEMESSAGE(message);
9:       DELETEVERTEX(successor.id);
10:      DELETEEDGE(vertex.id, successor.id);
11:    end for
12:  end if
13: end procedure
```

shows the final graph structure after two rounds of compression. The compression rounds (superstep 2-5) are similar to Figure 4.2 and not shown here.

4.3.6 Bubble Removal and Contig Generation

Algorithm-6 and Figure 4.4 illustrates the steps involved in one round of GiGA's bubble removal process. A parameter called *max_bubble_len* is set to $5k$. Vertices with length less than *max_bubble_len*, and having in-degree as well as out-degree of one, are considered as potential bubbles. In the first superstep, every vertex matching this criterion sends a message to their immediate predecessor containing its own value and the frequency. In the second superstep, the vertices which received the messages from their successors compute the dissimilarity between the received k -mers from the successor-vertices using a Levenshtein-like edit-distance algorithm which returns the minimum number of characters that can be changed to equalize both the vertices. If the dissimilarity between the vertices is within a threshold, then one of the vertices is a bubble. Then, the length of the vertices and the frequency associated with both are compared. If both the vertices have the same length, the one with the lower frequency is considered as an erroneous vertex and is purged from the graph. Like tip removal, removal of the bubbles will again generate some new linear paths in the graph that is compressed again. Figure 4.4 illustrates an example. In superstep 0, Vertex $F'G'H'I'$ and $FGHI$ are identified as a potential bubble and they send a message to their

E containing as discussed above. In superstep 1, vertex G removes $F'G'H'I'$ from the graph as its frequency is less. Superstep 5 shows the final graph structure after two rounds (Superstep 2-5) of compression which are similar to Figure 4.2.

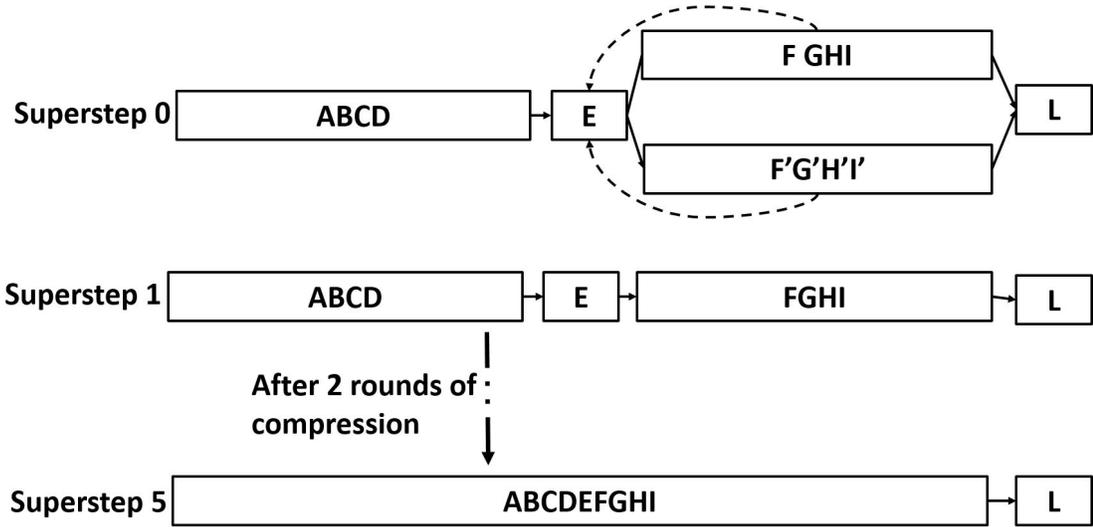


Figure 4.4: Bubble removal: Each two supersteps make a round. Dotted lines show the messages.

4.4 Evaluation

4.4.1 Assembly Quality Assesment

GiGA’s assembly quality is assessed by comparing it with two other assemblers, ABySS and Contrail using the datasets in Table 4.1. The quality is compared in terms of Corrected NG50 and missassembly using the GAGE benchmark datasets (first three in Table 4.1) for which the reference genome is available. Then GiGA’s scalability is demonstrated using the human chromosome (HCR-14) dataset. Finally, a large Yoruban male genome dataset is assembled to demonstrate the efficiency of GiGA when assembling large-scale genome over hundreds of coes.

Algorithm 6 Bubble Removal

```

1: procedure COMPUTEPARALLELFORALLVERTICES(superstepID, messages)
2:   if (superstepID is even) then
3:     if (Length of vertex.contig is less than equal to  $5k$  and vertex has single incoming and outgoing edge
         and the length of vertex.contig is less than  $5k$ ) then
4:       SENDMESSAGE(predecessor, {vertex.id, vertex.contig, vertex.frequency, vertex.neighbors});
5:     end if
6:   else
7:     for (each message in messages) do
8:       successors[i].{successor.id, successor.contig} ← PARSEMESSAGE(message);
9:       i ++;
10:    end for
11:    selectedSuccessor ← successors[0]
12:    for (each successor in successors) do
13:      dist ← get the Levenshtein distance between selectedSuccessor.contig and successor.contig
14:      if (dist less than a predefined threshold) then
15:        if (successor.frequency is less than or equals selectedSuccessor.frequency) then
16:          DELETEVERTEX(successor.id);
17:          DELETEEDGE(vertex.id, successor.id);
18:          ADDEDGE(vertex.id, successor.neighbors);
19:        else
20:          DELETEVERTEX(selectedSuccessor.id);
21:          DELETEEDGE(vertex.id, selectedSuccessor.id);
22:          ADDEDGE(vertex.id, selectedSuccessor.neighbors);
23:          selectedSuccessor ← successor
24:        end if
25:      end if
26:    end for
27:  end if
28: end procedure

```

Table 4.1: Datasets

	<i>S. aureus</i>	<i>R. sphaeroides</i>	HCR-14	<i>E. coli</i>	Yoruban Male
Source	GAGE	GAGE	GAGE	SRX000429	SRA000271
Read Size(bp)	255×10^6	410×10^6	5.9×10^9	749×10^6	141.5×10^9
Read Length (bp)	37 and 101	101	101	36	101
Total reads	4,791,974	4,105,236	59,414,772	10,408,224	2billion
Ref. Genome size	2,872,915	4,603,060	88,289,540	4.7×10^6	3.3×10^9
Dataset size (gigabytes)	0.3	0.6	10.0	3.2	452.0

1) Corrected NG50: It is calculated in two steps. First, the contigs are broken at each error compared with the reference genome. Then, NG50 is computed based upon the true reference genome size. If true reference genome is not available, the first step is omitted and calculated the NG50 based upon the estimated reference genome size. The GAGE script is used in their website¹ to calculate this.

¹<http://gage.cbcb.umd.edu/results/index.html>

2) Misassembled Contigs: It is the total number of contigs that contain different misassembly events, such as, relocations, translocations, and inversions. QAST [56] is used to calculate this.

4.4.2 Assembly of GAGE Benchmark Datasets

This work uses the first three datasets in Table 4.1 that are openly available on the GAGE website, *Staphylococcus aureus*, *Rhodobactor sphaeroides* and human chromosome, for benchmarking the performance of the assembler. These three datasets are used for two distinct reasons. First, these three sample datasets were previously assembled using conventional Sanger technology, and the finished reference genomes are also available on the same GAGE website. Having finished genomes enables us to evaluate the correctness of the assembler. Second, these three datasets have been assembled before using several other assemblers during GAGE study. Table 4.2, 4.3 and 4.4 shows the accuracy of GiGA compared to ABySS and Contrail over GAGE datasets (*S. aureus*, *R. sphaeroides* and HCR14 respectively). Assembled genomes for ABySS is downloaded from GAGE website. It can be easily observed that GIGA shows substantially lower misassembly and higher NG50 comparing to ABySS. GiGA’s assembly quality is almost comparable to Contrail in terms of accuracy. However, GiGA performs a magnitude faster than Contrail that is demonstrated in the subsequent sections.

4.4.3 Assembly of *E. coli*

E. coli dataset consists of 10.4 million paired-end, 36 bp Illumina reads with (NCBI Short Read Archive, accession no. SRX000429). The k -mer size of 27 is used for the evaluation. The performance comparisons are shown in Table 4.5. The quality of assembly in terms of NG50 length is better than ABySS, and is comparable to Contrail. In terms of the execution time, the overall assembly was completed in 32.3 minutes i.e. 1.7x faster than Contrail.

Table 4.2: Accuracy of *S. aureus*

	GiGA	ABySS	Contrail
#Contigs	298	300	309
Corrected NG50	25725	241819	25200
NG50 count	34	35	30
Max contig	96737	125049	96737
Misassembled contigs	0	4	0

Table 4.3: Accuracy of *R. sphaeroides*

	GiGA	ABySS	Contrail
#Contigs	737	1912	309
Corrected NG50	10804	4215	11718
NG50 count	134	283	126
Max contig	65538	54734	51683
Misassembled contigs	1	78	1

4.4.4 Scalability of GiGA

Figure 4.5 shows the performance of GiGA in terms of computational time and scalability. All the experiments are performed in a cluster where each node has 16 processing cores, 500GB hard disk and 32GB memory (RAM). All nodes of the cluster are connected through a 40Gbps QDR Infiniband switch with a blocking ratio of 2:1. Because of many jobs running simultaneously on the cluster, the effective bandwidth between any two nodes were 950Mbps. GiGA’s scalability is shown by assembling a Human chromosome (59.5 million reads), which is the

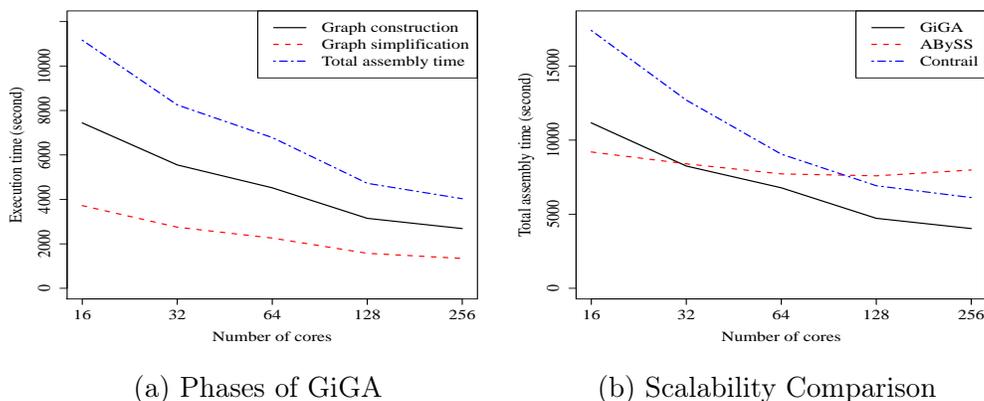


Figure 4.5: Scalability result

Table 4.4: Accuracy of HCR 14

	GiGA	ABySS	Contrail
#Contigs	76049	51790	76209
Corrected NG50	658	1269	700
NG50 count	33271	16643	34223
Max contig	19446	30053	19321
Misassembled contigs	3	17	3

Table 4.5: Assembly of *E. coli* using $k = 27$

	GiGA	ABySS	Contrail
No.of contigs	234	251	273
NG50	191,103	96,308	119,782
Max	237, 843	268,283	236,834
Cores	16	16	16
Time (Minutes)	32.3	34.5	55.2

largest among the three GAGE benchmark datasets used in this work. The size of this dataset is 10-gigabytes in fastq format which produces 20GB of graph.

Figure 4.5a shows almost linear scalability of both GiGA’s graph construction and simplification phases. Figure 4.5b compares the scalability of GiGA with ABYSS and Contrail in terms of total assembly time. It is worthy to notice that with fewer cores the performance of GiGA is comparable to ABYSS. However, after a certain point with increase in the number of cores the performance of ABYSS degrades due to network bottleneck. Since GiGA considers data locality, it continues to speed up with the increase in the number of cores. Comparing the best execution time, GiGA shows almost 2x better performance than ABYSS. On the other hand, even if Contrail shows good scalability because of data locality, GiGA always shows almost 1.5x speedup because of its in-memory graph processing comparing to Contrail’s disk-based approach.

4.4.5 Assembly of Human Genome

To demonstrate GiGA’s capability to assemble larger genomes, a Yoruban male genome dataset (Accession #SRA000271) is used. This dataset has read length of

101 with 47X coverage. The size of the dataset is 452-gigabytes which produces almost 3.5-terabytes of de Bruijn graph. Table 4.6 shows GiGA’s assembly result for this dataset using k -mer size of 57. The entire assembly was completed in 8.5 hours over 512 cores with a high NG50 of 827. Due to time and resource limitation on the computation cluster, ABySS could not run over 512 cores possibly because of high communication overhead. On the other hand, Contrail, being Hadoop-based, took an extremely long time and, finally, ran out of the storage space. Hence, a fair comparison could not be done with these assemblers and GiGA’s performance metrics are reported only.

Table 4.6: Assembly of Yoruban male genome using $k = 57$

	GiGA
No.of contigs	3,032,297
NG50	827
Max	35,465
Cores	512
Time (Hours)	8.5

4.5 Conclusion

This work introduces GiGA, a Parallel Giraph Based Genome Assembler that is developed to address the challenges involved in large-scale genome assembly, which recently made its way to the forefront of big data challenges. Scalability is one of the major components of next generation large-scale genome assembly software tools that is focused on this work.

GiGA maps the entire de Bruijn graph simplification process to Valiant’s BSP model and develops the assembler using Giraph, a state of the art, large-scale graph processing framework developed atop Hadoop which can scale over large commodity cluster. The evaluation over several datasets, ranging from small bacterial genomes to a large human genome and GAGE benchmark data, shows that GiGA achieves significantly improved performance and scalability over hundreds of cores with fewer misassemblies compared to other parallel assemblers.

Chapter 5

Parallel Long-read Error Correction with Hadoop

The third generation sequencing platforms have been emerged with a promise of better and less fragmented assembly compared to the second generation sequencing platforms because of the substantial rise in the read-length. However, the sequencing is costly and have high error rate ($>15\%$). So, an accurate yet low-cost error correction tool is of paramount need.

Motivated by this, this chapter developed ParLECH, a distributed, scalable, cost-effective, de novo, hybrid error correction software tool for PacBio long reads. ParLECH leverages the low error rate (1%), low cost and deep coverage of Illumina short read sequences. ParLECH uses the power of MapReduce and distributed NoSQL to analyze the high throughput Illumina reads in real time. For accuracy, ParLECH utilizes the k -mer coverage information of Illumina sequences. This chapter developed a distributed version of Widest-Path traversal algorithm which maximizes the minimum k -mer coverage in a path of de Bruijn graph constructed from the Illumina short reads. This widest path replaces the corresponding error region in a PacBio long read.

ParLECH can handle hundreds of gigabytes (GB) of data with almost linear scalability and high accuracy. This chapter demonstrates the data handling capability by processing a large human genome of 350GB over 128 nodes in 28.6 hours. On the other hand, ParLECH aligned almost 92% PacBio bases of an *E. coli* sequence with the reference genome proving its accuracy.

5.1 Introduction

The rapid development of genome sequencing technologies has become the major driving force for genomic discoveries. The second generation sequencing technologies (e.g., Illumina) provided the researchers with the required throughput

at significantly lower cost (in the range of \$41 to \$502 per giga base pair [57]) that enabled the discovery of many new species and variants. However, the short length (a few hundred base pairs only) of reads relative to the repeat sequences resulted in a fragmented assembly with thousands of short contigs hindering many of the downstream applications such as genome finishing, gap filling in scaffolds, more complete genome assembly, etc.

To address the issues with short read length, third generation sequencing technologies (e.g., PacBio, Oxford Nanopore, etc.) started emerging recently. By producing long reads greater than 10kbp these third generation sequencing platforms provide the researchers with significantly less fragmented assembly with the promise of a much better downstream analysis.

However, the production of these long reads is costly (\$2000 per giga base pair [57]) and severely constrained by their higher error rate. For example, on an average, a PacBio sequencing machine produces 15% of error in contrast to only 1% in an Illumina sequencing machine. Hence, an accurate error correction tool for these long reads can be extremely helpful for more complete analysis of the genome of different species.

This work proposes ParLECH, a distributed, scalable, cost-effective, hybrid solution for correcting PacBio reads. ParLECH leverages the low error and high coverage of the Illumina reads to rectify the PacBio reads. Consequently, the Hybrid error correction reduces the coverage requirement for long reads reducing a significant amount of cost.

ParLECH first constructs a de Bruijn graph (DBG) out of the Illumina short read sequences and then uses the k -mer coverage information of Illumina sequences to correct the long PacBio sequences. This approach is similar to other long read error correction tools such as, LoRDEC [58] and Jabba [59]. However, unique to ParLEC, an algorithm to find the widest path is developed which maximizes the

minimum k -mer coverage between a source and a destination node in the DBG. The algorithm corrects the long read with significantly higher accuracy. For an E. Coli genome, ParLEC aligned 92% of the PacBio base pairs correctly comparing to 86% in case of LoRDEC.

This chapter develops the error correction algorithm using Hadoop MapReduce and a distributed NoSQL called Hazelcast. Consequently, ParLEC is able to scale with terabytes of sequencing data over hundreds of compute nodes. Scalability is critical for hybrid error correction because of the involvement of high throughput Illumina data which may grow up to terabytes for large and complex genomes. ParLECH's data handling capability is demonstrated by correcting 350GB (GigaByte) of PacBio sequences from human genome by leveraging the lower error rate of 452GB of Illumina sequence (64x Coverage) over 128nodes in 28.6 hours. Existing tools such as, LoRDEC [58] cannot handle this huge amount of data.

Rest of the paper is organized as follows: Section 5.2 discusses the related work to our current effort. Section 5.3 describes the error correction procedure of ParLEC. Section 5.4 describes the results and compare ParLEC with existing long read error correction tools in terms of both accuracy and execution speed. Finally, Section 5.5 concludes the paper.

5.2 Related Work

Second generation sequencing platform such as, Illumina produces short reads at an error rate of 1-2% [22]. However, most of the errors are substitutions. Consequently, the low cost of production of high coverage data enabled self correction of errors without using any reference. Utilizing the basic fact that the k -mers resulting from an error base will have significantly lower coverage compared to the actual k -mers, many error correction tool have been proposed. Quake [24], Reptile [26] and Hammer [30], RACER [31], Coral [60], Lighter [34], Musket [35], Shrec [32], DecGPU [36], Echo [25], ParSECH [21] etc. are to name a few.

Third generation sequencing platform such as, PacBio, on the other hand, produces long reads at an error rate of 10-15% [23] which is significantly higher compared to the Illumina sequences. Furthermore, the error model is mostly indel prohibiting the use of the error correction tools mentioned earlier which worked well for substitution errors of second-generation reads. Complicating the scenario, the PacBio sequences incur almost 10 times more production-cost compared to that of the Illumina sequences putting a practical barrier in the coverage requirement for self-error correction of PacBio reads.

LorMA [61] is a self-correcting tool which needs almost 50x coverage. A more efficient methodology for self-correction is proposed in Canu [62] using a tf-idf hash of reads reducing the coverage requirement by almost a half. However, considering 10 times more cost of PacBio data still poses a severe bottleneck on its practical use especially for complex, large genomes.

A much practical and cost-effective solution is proposed in [58, 59, 63, 64, 65, 66, 67] where the low-cost, high-quality Illumina short reads are used as a reference to correct the PacBio reads. LoRDEC [58] developed a de Bruijn graph-based methodology for correcting PacBio reads. After identifying the error region in a long read, LoRDEC performs a local assembly on the de Bruijn graph prepared from the Illumina reads to replace that region. Jabba [59] also uses a de Bruijn graph-based approach. However, it uses different size of k -mer iteratively to polish the unaligned regions of the long reads. Other hybrid tools adopted alignment-based approaches where the short reads are first mapped to the long reads to create an overlap graph followed by some consensus-based algorithm to rectify the PacBio errors. For example, PacBioToCA [64] and LSC [65] after alignment, call for a per base consensus to correct the errors. ColorMap [63] on the other hand, applies a Dijkstra's shortest path algorithm where each edge retains the information of consensual dissimilarity. Proovread [66] reaches the consensus by repeating the

alignment procedure in many iterations by incrementally increasing the sensitivity of the long reads.

Among these methodologies, de Bruijn graph-based algorithm of LoRDEC [58] performs a magnitude faster comparing to the other tools. ParLECH, therefore, follows a similar approach to leverage the performance promises of de Bruijn graph. Moving a step forward, ParLECH is developed as distributed so that it can scale over hundreds of compute nodes over terabytes of data. Furthermore, to improve the accuracy ParLECH developed a distributed version of widest path traversal algorithm. Unlike LoRDEC’s algorithm, it leverages the k -mer coverage information of the short reads during the local assembly also which improves ParLECH’s accuracy.

5.3 Methodology

5.3.1 Overview

Figure 5.1 shows the overview of ParLECH’s error correction approach. The hybrid approach for error correction is inspired by de Bruijn graph-based approach of LoRDEC. This approach leads to significant performance gain in terms of execution time. However, for improved accuracy, an algorithm (refer to Algorithm 7) to calculate the widest path is implemented for DBG traversal. For better scalability, The algorithm is developed using Hadoop so that it can be distributed over hundreds of compute nodes.

To correct a long read of PacBio, ParLEC first constructs a de Bruijn graph from the short reads of Illumina maintaining the coverage information of each k -mer in the graph. Then it partitions a long read into weak and solid regions (lines and rectangles in Figure 5.1) according to the k -mer coverage in short reads. To correct the errors ParLEC then selects the k -mers around a weak that serve as source and target nodes in the DBG. Then it follows a widest path algorithm

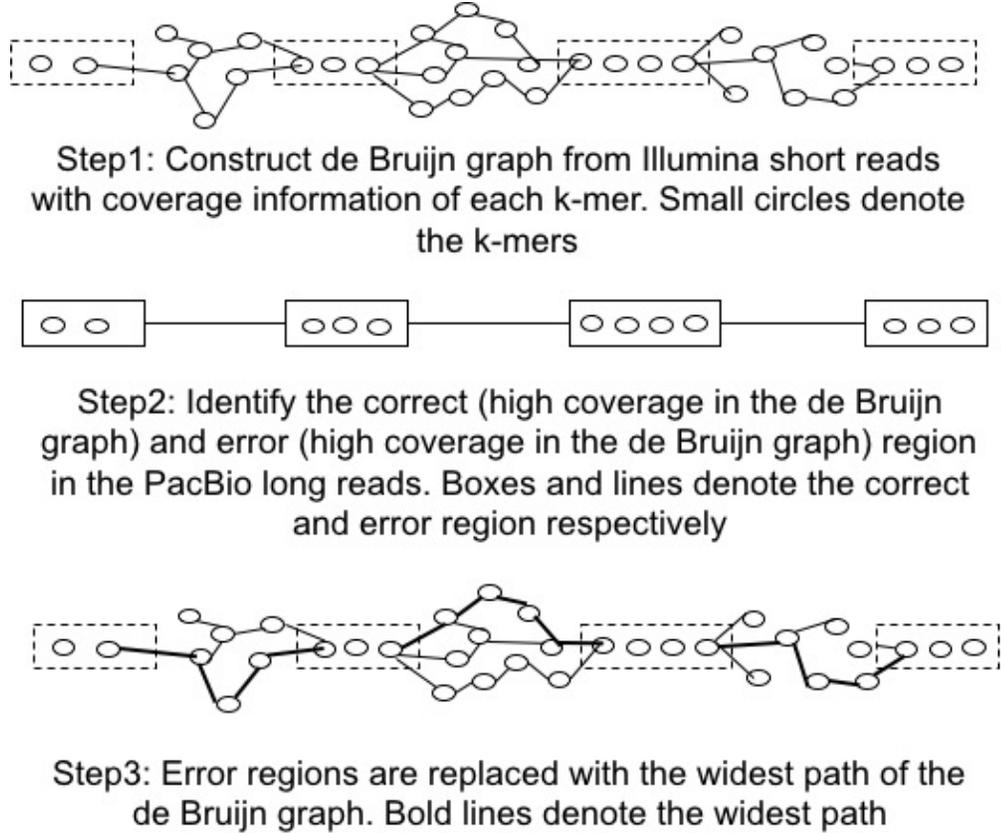


Figure 5.1: Error correction steps

between the source and target k -mer in the DBG which maximizes the minimum coverage of the k -mers (Algorithm 7). Once the path in the DBG is found, the weak region in the long read is replaced with that path.

5.3.2 Error Model

Since ParLECH leverages the lower error rate of Illumina reads to correct the PacBio sequencing errors, let us first describe an error model for Illumina sequences and its consequence on the DBG constructed from these reads. Let us consider two reads R_1 and R_2 representing the same region of the genome and R_1 has one error base. Let us assume the k -mers between the position pos_{begin} and pos_{end} is an error region in R_1 where error base is at position $pos_{error} = \frac{pos_{end} + pos_{begin}}{2}$.

Claim 1: Coverage of at least one k -mer in the region between pos_{begin} and pos_{end} in R_1 is lower than the coverage of any k -mer in the same region of R_2

Proof of Claim 1: There are k k -mers between pos_{begin} and pos_{end} in R_1 and their coverage is independent identically distributed (iid) random variables. Hence, the minimum of all the occurrences of those k k -mers can be expressed as

$$Y = \min(x_1, x_2, x_3, \dots, x_n) \quad (5.1)$$

The corresponding distribution i.e., the minimum of the k -mers is

$$P(Y \leq x) = 1 - (1 - F(x))^n \quad (5.2)$$

Where, $F(x)$ is the distribution of coverage of the k k -mers.

Theoretically, the k -mer coverage should follow exponential distribution [24]. Let us assume, $F(x)$ follows an exponential distribution with rate r i.e. $F(x) = 1 - \exp(-rx)$,

$$P(Y \leq x) = 1 - \exp(-\mu x) \quad (5.3)$$

Where, $\mu = rk$.

For a carefully chosen cut off point (as discussed in [68] and [24]) where the ratio of error k -mers to true k -mers is high the value of r at R_1 is significantly higher than that in R_2 . That means the probability of having the k -mer with minimum coverage is higher in R_1 comparing to R_2 which proves the theorem. Although the reads R_1 and R_2 are assumed to represent the same region in the genome for the sake of convenience and easy understanding, the theorem can be easily generalized for any reads with overlap where one of the reads has the error base in the overlapped region.

5.3.3 Choosing the Right Path in De Bruijn Graph

From the short reads a de Bruijn graph is constructed where each vertex represents a k -mer. An edge (u, v) is added between two vertices u and v if there is a $k-1$

suffix-prefix overlap between u and v and weight of the edge, $w(u, v) = coverage(v)$.

A de Bruijn graph constructed this way from reads R_1 and R_2 will always have a fork structure leading to two different paths (refer to Figure 5.2). According to Claim 1, the error path has significantly high probability of the k -mer with minimum coverage introducing a coverage-bottleneck in that path. Hence, we construct a widest path algorithm (refer to Algorithm 7) which always maximize the minimum k -mer coverage in a path in a de Bruijn graph.

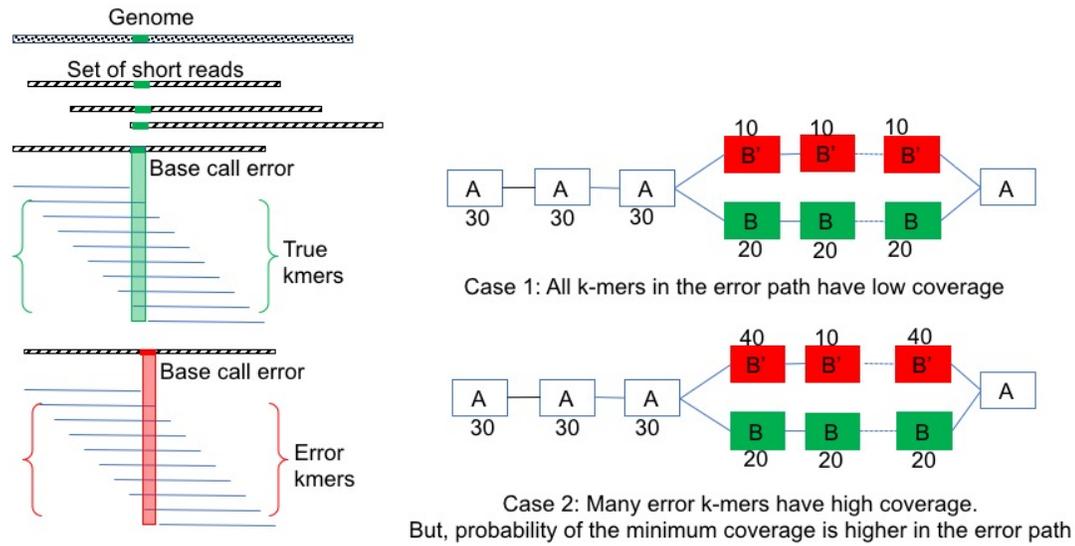


Figure 5.2: Widest path algorithm has higher probability to produce correct result even when high coverage k -mers are present in the error path

5.3.4 ParLECH's Distributed Architecture

ParLECH's distributed architecture is inspired by ParSECH as discussed in Chapter 3. Like ParSECH, ParLECH also uses Hadoop and Hazelcast. In ParLECH, Hadoop is used for MapReduce-based programming and Hazelcast is used as an in-memory distributed storage for the de Bruijn graph along with the coverage information of each k -mer. As discussed in Chapter 3 Hadoop in conjunction with Hazelcast can enable hundreds of thousands of searches in parallel depending on availability of cores, thus makes the k -mer searching and graph traversal process for error correction fast and scalable.

Figure 5.3 shows the distributed architecture and the error correction pipeline of ParLECH. It has three different phases: 1) count k -mers, 2) locate errors, and 3) correct errors. The HDFS is used to store the raw short and long read sequences which act as input to ParLECH. Whereas, Hazelcast is used to store the de Bruijn graph in memory in a distributed way. Each phase of ParLECH consists of a parallel and distributed Hadoop MapReduce (or Map-only) jobs that do the required computation for that phase. In the first phase, Hadoop is used to construct the de Bruijn graph from the Illumina short read sequence. In the subsequent phases, both Hadoop and Hazelcast are used to locate and correct the errors in the PacBio reads. Finally, the corrected reads are written on the HDFS.

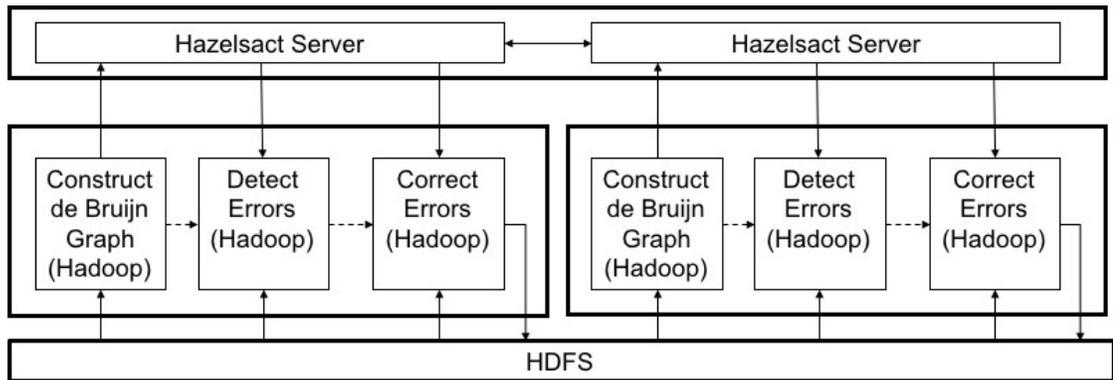


Figure 5.3: ParLECH's distributed architecture and error correction pipeline

5.3.5 De Bruijn Graph Construction

Constructing the de Bruijn graph with Hadoop is straightforward. Figure 5.4 shows an example construction of a de Bruijn graph from two short reads. The process is almost similar as described in Chapter 4. However, instead of only outgoing edges from a vertex, the incoming edges of the vertex is also stored. In the map phase of the Hadoop-based algorithm, each read is divided into several short fragments of length k , known as a k -mer. Three subsequent k -mers are emitted as key-value pairs, where the second one (key) represents a vertex in the graph, the first one represents an incoming edge to the key, and the third one

represents an outgoing edge from the key. Both first and third k -mers act as value for the key k -mer (i.e., the second one). Additionally, the value field also contains an integer (1) which corresponds to the coverage of the key. After the map function completes, the shuffle phase partitions the intermediate key-value pairs on the basis of the keys to collect the edges and the count of occurrences from the same source k -mer. Finally, the reduce function aggregates the incoming edges and outgoing edges separately and then sum up the count of occurrences of each source k -mer. ParLEC then saves the graph structure in the form of a distributed hash table in memory using Hazelcast NoSQL where the k -mer acts as the key and its coverage and the set of outgoing edges both act as the value.

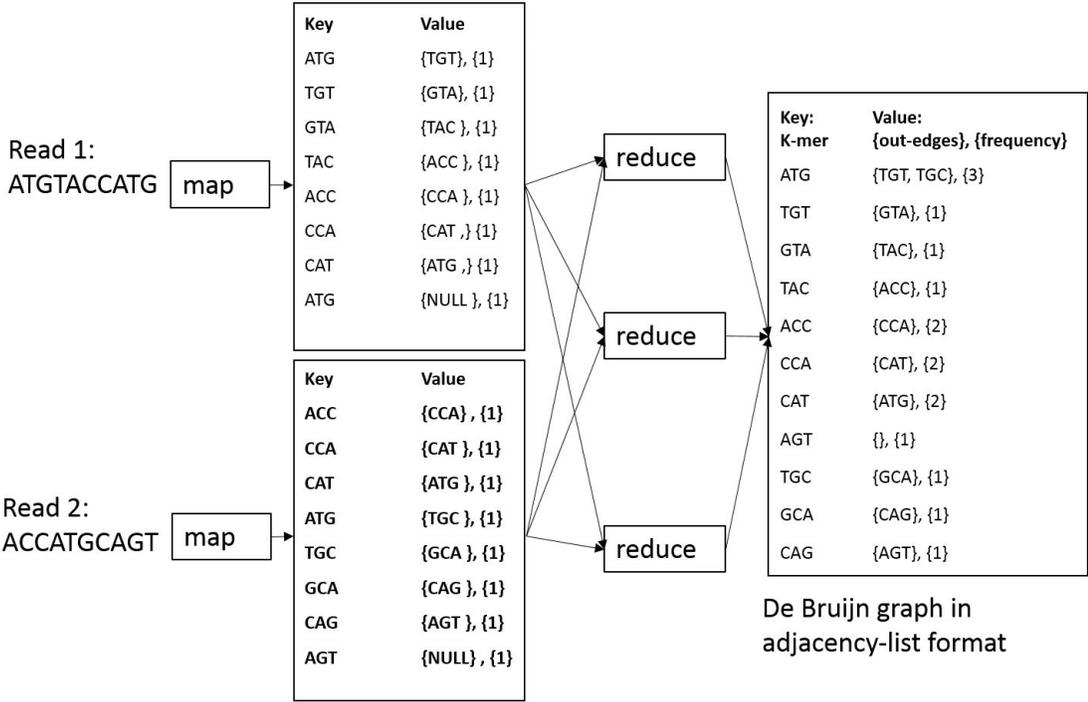


Figure 5.4: De Bruijn graph construction using Hadoop.

5.3.6 Locating Errors in Long Read

This is a Hadoop map-only job which is distributed over multiple nodes. In this phase, the map task scans each long read and constructs k -mer of the same length

as in case of the short reads. Then, the coverage information of the short read is used to detect the errors in the long reads. Each k -mer generated from the long read is queried in Hazelcast. If its coverage is greater than a predefined threshold the k -mer is marked as strong. Otherwise, the k -mer is marked as weak. If it is marked as weak, the k -mer is subjected to correction. If two or more consecutive k -mers have coverage less than the threshold, the entire region in the long read is marked as weak and is subjected to correction.

5.3.7 Correcting Errors in the Middle of Long Reads

Algorithm 7 Widest Path

```

1: procedure MODIFIEDDIJKSTRA(Graph, source, destination)
2:   for (each vertex  $v$  in Graph) do
3:     width[ $v$ ] := -infinity ;
4:     previous[ $v$ ] := undefined ;
5:   end for
6:   width[source] := infinity ;
7:    $Q$  := the set of all nodes in Graph ;
8:   while ( $Q$  is not empty:) do
9:      $u$  := vertex in  $Q$  with largest width in width[] ;
10:    remove  $u$  from  $Q$  ;
11:    if (width[ $u$ ] = -infinity) then
12:      break ;
13:    end if
14:    for (each neighbor  $v$  of  $u$ ) do
15:       $alt$  := max(width[ $v$ ], min(width[ $u$ ], width_between( $u$ ,  $v$ ))) ;
16:      if  $alt$  > width[ $v$ ]: then
17:        width[ $v$ ] :=  $alt$  ;
18:        previous[ $v$ ] :=  $u$  ;
19:      end if
20:    end for
21:  end while
22: end procedure

```

Like locating errors, correction is also a Hadoop map-only job distributed over multiple nodes. A weak region is bordered by a set of solid k -mers on each side. The map task scans each read and takes as input the source and target solid k -mers, the region sequence. Like LorDEC, solid k -mers serve as source and target nodes in the DBG. Any path between these nodes encodes a sequence that can be assembled from the short reads. An algorithm to find the widest path in a graph is developed which maximizes the minimum k -mer coverage of a path in the DBG. As discussed in Claim 1, the error path has the highest probability to contain the k -mer with the minimum coverage. As shown in Figure 5.2 Practically it means

that even if there are some error k -mers present with high coverage, there will be at least a single k -mer whose coverage will create a bottleneck in the path. Hence, the widest path algorithm selects the optimal solution that is, the correct sequence between the source and the target k -mers with significantly high probability.

Algorithm 7 shows the widest path algorithm developed by slightly modifying Dijkstra's shortest path algorithm with a time complexity of $O(E \log V)$.

Proof of correctness of Algorithm 7: Assume that at any point, let \mathcal{S} be the set of the vertices to which the widest path from s has been found. We prove the correctness using induction on the size of \mathcal{S} . The base-case is when $|\mathcal{S}| = 1$, i.e., when s is added to \mathcal{S} , and the correctness is obvious. Inductively assume that for any vertex $u \in \mathcal{S}$, we have $width[u] = \delta(s, u)$. Clearly, when we add the next vertex, say v , it suffices to show that $width[v] = \delta(s, v)$.

Assume the contrary, i.e., $\delta(s, v) > width[v]$. (Obviously, $width[v] > \delta(s, v)$ does not hold as we have found at least as wide a path.) Suppose, the widest path from s to v is P , which means

$$minEdgeWeight(P) = \delta(s, v) > width[v]$$

Note that P must leave \mathcal{S} via an edge from x to y where $x \in \mathcal{S}$ and $y \notin \mathcal{S}$. Due to the algorithm, we have $width[y] \geq \min\{width[x], weight(x, y)\}$, which implies

$$width[y] \geq \min\{width[x], weight(x, y)\} \geq minEdgeWeight(P)$$

Finally, since the algorithm selects the largest-width vertex, and v is selected at this point, we get

$$width[v] \geq width[y]$$

By combining the inequalities, we get

$$width[v] \geq width[y] \geq minEdgeWeight(P) > width[v]$$

Hence, the initial assumption is false, and the correctness follows.

5.3.8 Correcting Error at the End of Long Read

If a weak region is detected at the end of a PacBio read, the strong, boundary k -mer in the DBG is searched. If it is the part of a chain structure of vertices (i.e., a path where each vertex has exactly one incoming and one outgoing edge), the entire chain after the vertex corresponding to the boundary k -mer is traversed until a fork structure is detected. Then the weak region of the PacBio read is replaced with that path of the DBG. Similarly, if the weak region is detected at the beginning of the PacBio read, the chain structure is traversed before the vertex corresponding to the boundary k -mer and replace the weak region with that path. If the k -mer is not the part of a chain structure the weak region in the PacBio read is discarded.

5.4 Evaluation

5.4.1 Dataset

To evaluate ParLECH, four different PacBio datasets are used as shown in Table 5.1. All the datasets are real. The corresponding Illumina datasets are shown in Table 5.2. The first three (i.e., *E. coli*, Yeast and Fruit fly) are relatively smaller dataset and is used to compare the accuracy of ParLECH with LoRDEC. The third one (i.e., Fruitfly) is relatively larger comparing to the first two (i.e., *E. coli* and Yeast). This dataset is used to analyze different metrics related to scalability and execution speed. The fourth one, a large human genome dataset is mainly used to showcase the data handling and scaling capability of ParLECH over hundreds of gigabytes of data over hundreds of nodes.

Table 5.1: PacBio dataset

PacBio Data	Accn. #	#Reads	Size	#Read length	#Reads aligned
<i>E. coli</i>	DevNet	1129576	1.032	1120	78.97
Yeast	DevNet	2315594	0.53	5874	82.12
Fruit fly	Bergman Lab	6701498	55	4328	51.14
Human	DevNet	23897260	312	6587	72.13

Table 5.2: Illumina dataset

Illumina Data	Accn. #	#Reads	Size	#Read length	#Reads aligned
<i>E. coli</i>	ERR022075	45440200	13.50	101	99.44
Yeast	SRR567755	4503422	1.2	101	93.75
Fruit fly	ERX645969	179363706	59	101	95.56
Human	SRX016231	1420689270	452	101	79.60

5.4.2 Computing Environment

For all the evaluations, LSU’s HPC cluster called SuperMic is used. Table 5.3 shows the cluster configuration. For any single job a maximum of 128 nodes are available. Each of the node has 20 cores, 64 GB of DRAM and one hard disk drive (HDD) with a capacity of 250GB. All the nodes are connected with a 56Gbps InfiniBand network with 2:1 blocking ratio. It should be noted that ParLECH’s is bottlenecked by the I/O throughput of each node of as it uses Hadoop’s disk-based computing to save costly DRAM. The performance can be significantly improved using multiple disks per node or deploying solid state drive (SSD). The analysis can be seen in later in the thesis in Chapter 6 and 7.

Table 5.3: Compute environment

Maximum #nodes	128
Processor	Intel IvyBridge Xeon
#cores per node	20
DRAM per node	64GB
Storage per node	250GB
Type of storage	Hard dis drive (HDD)
Network	56Gbps InfiniBand

5.4.3 Accuracy Metrics

The major accuracy metrics that have been used are as follows:

1) %Reads and base pair aligned: To check the accuracy of ParLECH, it is investigated that how well the corrected long reads and the base pairs aligned to the reference genome. The percentage of base pair aligned successfully to the reference genome indicates the ratio of the total number of bases that are aligned successfully to the total number of base pair contained in the original dataset. To align the E. Coli, Yeast and Fruit fly dataset to their corresponding reference genome, BLASR [69] is used as it tends to bridge the long indels better and thus reports longer alignments. However, for large human genome BWA-mem [70] is used which produces the result faster.

2) Gain: After alignment, further details of each of the corrected regions is measured in terms of gain which indicates the fraction of errors effectively corrected from the genome dataset. It can be defined as follows,

$$Gain = \frac{TP - FP}{TP + FN} \quad (5.4)$$

where, TP (true-positive) indicates the number of errors which are successfully corrected, FP (false-positive) is the number of true bases which are changed wrongly, and FN (false-negative) is the number of errors which are falsely detected as correct.

5.4.4 Comparing Different Graph Traversal Algorithm

The widest path algorithm (ParLECH_{WP} or simply ParLECH) is first compared with two other graph traversal algorithms such as, Dijkstra's shortest path (ParLECH_{SP}) and a greedy traversal (ParLECH_{Greedy}) algorithm. Table 5.4 shows the comparison result of these three different algorithms.

1) ParLECH_{SP}: The Dijkstra's shortest path algorithm searches for the short-

Table 5.4: Different types of algorithms: Widest-Path (ParLECH_{WP}) vs Dijkstra’s shortest-path (ParLECH_{SP}) vs greedy algorithm (ParLECH_{Gr}).

Data	Methodology	%Aligned reads	%Aligned bases
<i>E. coli</i>	ParLECH _{WP}	93.69	92.15
	ParLECH _{SP}	87.55	86.49
	ParLECH _{Gr}	76.68	70.92
Yeast	ParLECH _{WP}	86.07	89.31
	ParLECH _{SP}	84.92	86.44
	ParLECH _{Gr}	75.77	74.68
Fruit fly	ParLECH _{WP}	65.92	62.42
	ParLECH _{SP}	54.53	49.41
	ParLECH _{Gr}	43.97	37.44

est distance path between two strong k -mers in order to bridge the weak regions (alternatively gap) between them in a PacBio read. The time complexity of this algorithm is similar to the widest path algorithm. However, the major drawback is that it cannot take the advantage of the k -mer coverage information. All the edges are assumed to have equal weight of 1. The widest path algorithm (ParLECH_{WP}) always produce better result comparing to the other two for all the three datasets.

2) ParLECH_{Gr}: The greedy algorithm, on the other hand, can take the advantages of the k -mer coverage. It is a variation of depth first search. While traversing the graph starting from a source, it selects the successor which has maximum coverage among all. However, this algorithm diverged among a huge number of alternatives ($O(4^n)$ where n is the number of vertices in the graph) in the de Bruijn graph and many times ended up in a tip of an entirely different path resulting in an exponential complexity including several backtracking and forward movement. Hence, to restrict its execution, a branching factor b is used such that after traversing b vertices successively in the graph starting from the source the algorithm backtracks if the destination vertex is not found. The algorithm aborts when all the successors of its current vertex are visited. Among all the three algorithms, this one (ParLECH_{Gr}) produces the worst result.

5.4.5 Comparison with LoRDEC

Table 5.5 compares the overall alignment accuracy and gain of ParLECH with LoRDEC. All the results are calculated on the basis of Blasr alignment. Since Blasr algorithm is embarrassingly parallel in nature i.e., it iterates the similar process over all the reads and generates the alignment statistics, multiple instances of Blasr are created on the subset of a bigger dataset and computed the results in parallel.

As it can be seen, ParLECH produces significantly better accuracy both in terms of alignment and gain over all the three datasets. Also, it can be observed that ParLECH’s result significantly depends on the length of the PacBio sequences. For *E. coli* and Yeast genome ParLECH produces significantly better result comparing to that of the fruit fly. The reason is that the fruit fly dataset has smaller read-length on average. In many reads, no strong k -mers have been found and those reads are not corrected like LoRDEC. However, unlike LorRDec, the strong k -mers in all the reads are checked even if its length is less than 5000 which are kept out of the computation in LoRDEC for better performance.

Table 5.5: ParLECH accuracy: ParLECH is more accurate than LoRDEC both in terms of alignment and gain

Data	Method	%Aligned reads	%Aligned bases	%Gain
E. Coli	Original	78.97	75.07	N/A
	ParLECH	93.69	92.15	90.05
	LoRDEC	87.55	86.49	87.15
Yeast	Original	82.12	88.69	N/A
	ParLECH	86.07	89.31	82.40
	LoRDEC	84.92	87.08	81.42
Fruiy fly	Original	51.14	46.04	N/A
	ParLECH	65.92	62.42	84.73
	LoRDEC	54.53	49.69	85.43

5.4.6 Scalability

Figure 5.5 shows the execution time and scalability of ParLECH. As shown in Figure 5.5a, LoRDEC outperformed ParLECH for E. Coli dataset on a single node because of ParLECH’s parallel computing overhead. However, ParLECH can

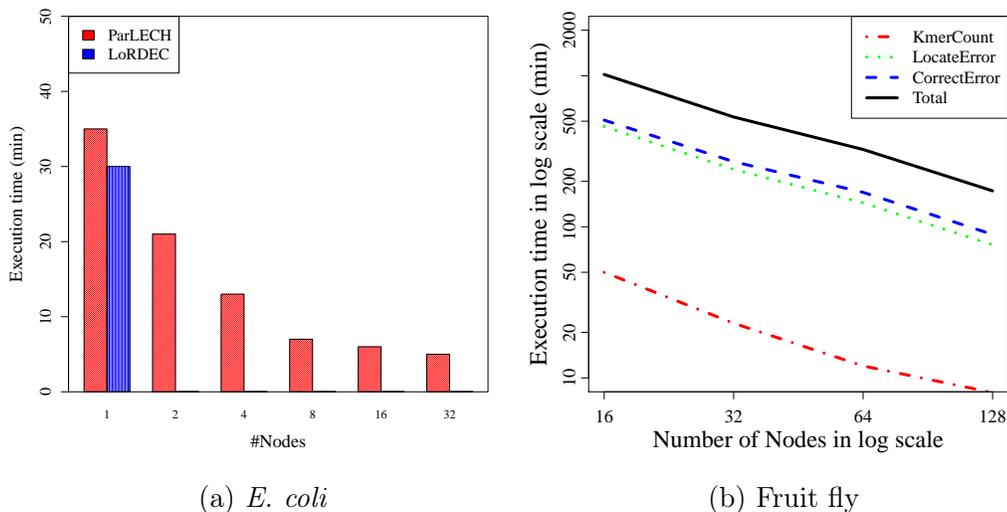


Figure 5.5: ParLECH scalability

be easily distributed over a Hadoop cluster many nodes. Consequently, ParLECH outperforms LoRDEC as soon as the computing load is distributed over many nodes.

On the contrary, LoRDEC’s algorithm, especially the DBG construction process cannot be distributed over multiple nodes. Though the error correction process works on each PacBio read independently, the software does not take care of any scheduling tool to distribute the load over multiple nodes leaving it to be done manually by the user incurring a significant amount of effort.

Figure 5.5b delves deeper in to the scalability property of ParLECH. As it can be seen, each phase of ParLECH i.e., DBG construction, error detection, and error correction scale almost linearly with increasing number of nodes. Consequently, the overall execution time of ParLECH shows almost linear scalability with increasing number of nodes.

5.4.7 Processing Large-scale Human Genome

To show the data handling capability of ParLECH a large human genome sequence dataset is processed. As shown in Table ??, this dataset consists of more

than 23million PacBio reads with an average length of 6587bp. The data size on disk is almost 312GB. The corresponding Illumina dataset has more than 14billion reads each of size 101bp yielding a total size of is 452GB on disk.

128 nodes have been used to process the data. The entire process took 28.6 hours in the computing environment as shown in Table 5.6. Table ?? shows the result. As it can be seen ParLEC aligned 78.3% of the read correctly to the reference genome and 75.43% bases correctly. In terms of gain, ParLECH shows 82.38% accuracy.

Table 5.6: Correcting human genome

PacBio data size	312GB
Illumina data size	452GB
#nodes used	128
Time	28.6 hours
%Read aligned	78.3
%Basepair aligned	75.43
Gain	82.38

5.5 Conclusion

This paper presents ParLECH, a scalable, fully distributed, sequencing error correction tool for PacBio sequences which utilizes the power of the current state-of-the-art big data analytic software tools, Hadoop and Hazelcast to scale with huge volume of sequencing data over hundreds of nodes. A widest path algorithm is also proposed for error correction which makes better use of the k -mer coverage information of Illumina read sequences to rectify the long PacBio reads.

The big data processing framework (based on Hadoop and Hazelcast) developed in this paper can ease the designing and rapid prototyping of embarrassingly parallel algorithms for massive scale data in other genomic applications. Unlike shared nothing architecture, the framework exposes a global tabular view of the entire genomic dataset to each computation unit (e.g., Hadoop worker) so that complex graph analysis and/or statistical analysis algorithms can be developed

easily over the entire set of data in an intuitive manner. The existing standalone codes can be easily parallelized with little or no modification with this framework. To this end, one of the future directions of this research definitely includes extending this framework for the large-scale genome analysis pipeline including genome assembly, variant calling, etc.

Chapter 6

Evaluating Different Distributed Cyberinfrastructure for Data and Compute Intensive Applications

Scientists are increasingly using the current state of the art big data analytic software tools for their data-intensive scientific applications over HPC environment. However, understanding and designing the hardware environment that these data- and compute-intensive applications require for good performance remain challenging. With this motivation, the thesis evaluated the performance of big data software over three different distributed-cyber-infrastructures, including a traditional HPC-cluster called SuperMikeII, a regular datacenter called SwatIII, and a novel MicroBrick-based hyperscale system called CeresII. The evaluation is done using GiGA, the large-scale genome assembler developed atop Hadoop and Giraph and discussed in Chapter 4.

To address the impact of both individual hardware components as well as overall organization, the configuration of the SwatIII cluster has been changed in different ways. Comparing the individual impact of different hardware components over different clusters, a 70% improvement in the Hadoop-workload has been observed and almost 35% improvement in the Giraph-workload in the SwatIII cluster over SuperMikeII has been observed by using SSD (thus, increasing the disk I/O rate) and scaling it up in terms of memory (which increases the caching). Then, the chapter provides significant insight on the efficient and cost-effective organization of these hardware components. In this part, the MicroBrick-based CeresII prototype shows similar of performance as SuperMikeII while giving more than 2-times improvement in performance/\$ in the entire benchmark test.

This chapter previously appeared in IEEE BigData 2015 [71]. Reprinted by permission.

6.1 Introduction

Since experimental facilities at large-scale sciences, such as astronomy, coastal science, biology, chemistry, physics, etc., have produced an unprecedented amount of data, scientific communities encounter new challenges, such as how to store data efficiently, how to process data optimally, etc. The fundamental model of computation involved in the scientific applications is rapidly changing in order to address these challenges. Deviating from the traditional compute-intensive programming paradigm, e.g., MPI, etc., many HPC applications have started using the current state of the art big data analytic software tools, such as Hadoop, Giraph, etc., for their data-intensive scientific workloads.

However, the traditional supercomputers, even with tera to peta FLOPs scale processing power, are found to yield lower performance than expected, especially because of the I/O- and memory-bound nature of the data-intensive applications. As a result, building efficient and cost-effective hardware infrastructure became more challenging. However, this started opening new opportunities for the hardware manufacturers. Furthermore, in the last few years, an increasing number of data-intensive HPC applications started shifting towards the pay as you go cloud infrastructure (e.g., Amazon Web Service, Penguin, R-HPC etc.) especially because of the elasticity of resources and reduced setup-time and cost.

As a consequence, there is a growing interest in all three communities, including HPC-scientists, hardware-manufacturers, as well as commercial cloud-service-providers, to develop cost-effective, high-performance testbeds that will drive the next generation scientific research involving a huge amount of big data. Also, millions of dollars are being spent in programs, such as XSEDE¹ and NSFCloud², where system designers and scientists from different academic organizations and

¹<https://www.xsede.org/>

²<https://www.chameleoncloud.org/nsf-cloud-workshop/>

manufacturing companies collaborated to address the challenges involved in developing novel distributed-cyber-infrastructures.

Despite this growing interest in both the scientific as well as the industrial community, there is a limited understanding of how the different types of hardware architectures impact the performance of these big data analytic software when applied to real-world data and compute-intensive scientific workloads. Therefore, it is critical to evaluate different types of distributed-cyber-infrastructure in the context of real-world, data-intensive, high performance, scientific workloads.

In this work, a large-scale de novo genome assembly is used as one of the most challenging and complex real-world examples of a high performance computing workload that recently made its way to the forefront of big data challenges [72] [73]. De novo genome assembly reconstructs the entire genome from fragmented parts called short reads when no reference genome is available. The assembly pipeline of the GiGA (Giraph-based Genome Assembler) involves a terabyte scale short read data analysis in a Hadoop job followed by a complex large-scale graph analysis with Giraph, thus, serving as a very good example of both data- as well as compute-intensive workload.

In this work, we present the performance result of PGA atop three different types of clusters as follows: 1) a traditional HPC cluster, called SuperMikeII (located at LSU, USA) that offers 382 computing nodes connected with a 40-Gbps InfiniBand, 2) a regular datacenter architecture, called SwatIII (located at Samsung, Korea) that has 128 nodes connected with 10-Gbps Ethernet and 3) a new MicroBrick-based prototype architecture, called CeresII that uses PCIe based communication (also located at Samsung, Korea).

The performance analysis is divided into two parts: Firstly, the individual impact of different hardware components over different clusters has been compared. There was almost 70% improvement in the data-intensive graph-construction stage

based on Hadoop and 35% improvement in the Giraph-based, memory-intensive graph-simplification stage in the SwatIII cluster over SuperMikeII by using SSD and scaling it up in terms of memory. SSD increases the disk I/O rate, thus reducing the I/O wait. Whereas, more memory increases the caching effect.

Secondly, the chapter provides significant insight on the efficient and cost-effective organization of different hardware components by modifying the underlying hardware organization of SwatIII cluster (the regular datacenter architecture) in many different ways to better understand the impact of different architectural balance. Here, the chapter provides significant insight on the cost-effective deployment of both a scaled out and a scaled up cluster, especially how to leverage SSDs in a cost-effective manner. In this part, the new MicroBrick-based prototype architecture, CeresII is found to provide almost similar performance as SuperMikeII while yielding almost 2-times improvement in performance per dollar.

The rest of the chapter is organized as follows: Section-6.2 describes the prior works related to the study. Section-6.3 defines the motivation of the study, that is, the issues in a traditional supercomputer to process the big data workloads with respect to Hadoop and Giraph. Section-6.4 describes the evaluation methodology where the chapter sheds light on the experimental testbeds, the workload and the input data that is used in this work. In Section-6.5, the performance result is presented by comparing the individual impact of different types of network, storage, and memory architectures over different clusters. Section-6.6 compares the performance of PGA over different types of hardware organizations. Finally, in Section-6.7 the chapter conclude this study.

6.2 Related Work

Earlier studies [74] [75], as well as the prior experiences [76], [77] show that state-of-the-art big data analytic software tools can be useful for HPC workloads involving huge amount of big data. Jha [75] nicely showed the convergence between

the two paradigms: the traditional HPC-software and the Apache Software Stack for big data analytic. As a consequence, a growing number of codes in several scientific areas, such as bioinformatics, geoscience, etc., are currently being written using Hadoop, Giraph, etc. Despite the growing popularity of using Hadoop and other software in its rich ecosystem for scientific-computing, there are very limited prior works that evaluated different distributed-cyber-infrastructures for these software tools when applied for data-intensive scientific workloads.

There are several performance analysis studies on using different types of hardware to accelerate the Hadoop job using the existing benchmark workloads. Vienne [78] evaluated the performance of Hadoop on different high speed interconnects such as 40GigE RoCE and InfiniBand FDR and found InfiniBand FDR, yields the best performance for HPC as well as cloud computing applications. Similarly, Yu [79] found an improved performance of Hadoop in traditional supercomputers due to high-speed networks.

Kang [80] compared the execution time of sort, join, WordCount, and DFSIO workloads using SSD and HDD and obtained better performance using SSD. Wu [81] found that Hadoop performance increases almost linearly with the increasing fraction of SSDs in the compute cluster using the TeraSort benchmark. They also showed that in an SSD-dominant cluster, Hadoop's performance is almost insensitive to different Hadoop performance parameters such as block-size and buffer-size. Moon, using the same TeraSort benchmark [82] showed a significant cost benefit by storing the intermediate Hadoop data in SSD, leaving HDDs to store Hadoop Distributed File System (HDFS) data. Li [83], Krish [84] and Tan [85] also reached the same conclusion as Moon [82] for other enterprise-level workloads such as, Hive queries, HBase enabled TPC-H queries etc.

All of the above studies have been performed either with existing benchmarks (e.g., HiBench [86]) or with enterprise-level analytic workloads, thus, they are

unable to address the HPC aspect of Hadoop. Furthermore, very limited studies consider the in-memory graph processing frameworks (e.g., Giraph, etc.) even though, graph analysis is a core part of many analytics workloads.

Many of the prior efforts, analyzed the impact of overall architecture on Hadoop Workload instead of analyzing the impact of a specific hardware module. Michael [87] investigated the performance characteristics of the scaled out and scaled up architecture for interactive queries and found better performance using a scaled out cluster. On the other hand, Appuswamy [88] reached an entirely different conclusion in their study. They observed a single scaled up server to perform better than an 8-nodes scaled out cluster for eleven different enterprise-level Hadoop workloads including log-processing, sorting, Mahout machine learning, etc. The study is significantly different in the following aspects. 1) Existing works focus on the data-intensive, enterprise-level Hadoop jobs (e.g., log-processing, query-processing, etc.). On the contrary, genome assembly is severely data- and compute-intensive. Additionally, it involves a large graph analysis which is extremely memory-intensive. 2) Existing works are limited in terms of job size. For example, the data size chosen in [88] can be accommodated in a single scaled up server. Such a restriction has not been put on storage space or memory. Consequently, the performance comparison is more generic and realistic. 3) Unlike the existing works, the thesis considers the genome assembly workflow instead of choosing a single job, thus, working closer to the real world.

6.3 Motivation: Issue in Running Big Data Applications on Traditional Supercomputers

"Traditional supercomputers focused on performing calculations at blazing speeds have fallen behind when it comes to sifting through huge amount of Big Data"[89]. This section briefly describes the programming model of two popular big data analytic software tools, Hadoop and Giraph. Then, it describes several

issues that are observed frequently in a traditional supercomputing environment while running the applications developed using these frameworks.

6.3.1 Programming Models for Big Data Analytic Software

Hadoop and Giraph were originated as the open-source counterpart of Google's MapReduce and Pregel respectively. Both the software tools read the input data from the underlying Hadoop Distributed File System (HDFS) in the form of disjoint sets or partitions of records. Then, the dataset undergoes a distributed computation following the MapReduce programming model. First, a map function defined by the user is applied to each record of each disjoint set simultaneously to extract some intermediate information from each record and written to the local file system in the form of key-value pairs. These intermediate key-value pairs are then grouped together on the basis of the unique keys and then, shuffled or hashed to the reducers. Finally, the reducer applies a reduce function (also defined by the user) to the value-list of each unique key. The final output is written to the HDFS. The MapReduce framework enables data- and compute-intensive applications to run large volume of distributed datasets over distributed compute nodes with local storage. On the other hand, Giraph uses the Bulk Synchronous Parallel model where computation proceeds in supersteps. In the first phase of a superstep, Giraph leverages Hadoop-mappers when a user-defined vertex-program is applied to all the vertices concurrently. In the end of each superstep, each vertex can send a message to other vertices to initiate the next superstep. Alternatively, each vertex can vote to halt. The computation stops when all the vertices vote to halt unanimously in the same superstep. Giraph enables memory- and compute-intensive applications to upload data into distributed memories over different compute nodes.

6.3.2 Network Issues

Traditional HPC clusters (e.g., SuperMikeII as shown in Table 6.1) use an InfiniBand interconnect with high bandwidth and low latency to deliver short size of messages. In addition, InfiniBand-based networks use a standard 2:1 blocking ratio because compute-intensive applications neither produce nor exchange much of data. However, Hadoop and Giraph were developed to work atop inexpensive clusters of commodity hardware based on Ethernet network to exchange large volume of data. Therefore, big data applications might suffer from bottleneck problems over HPC-clusters with typical high blocking ratio networks

For example, during the shuffle phase of a Hadoop job, there is a huge data movement across the cluster. However, in other phases, the data movement is minimal in the network when mappers and reducers carefully consider the data locality. On the other hand, Giraph is more network-intensive. At the end of each superstep a huge amount of messages are passed across all the Giraph workers. Furthermore, every pair of workers uses a dedicated communication path between them that results in an exponential growth in the number of TCP connections with the increase in the number of workers. At these points, the data network is a critical path, and its performance and latency directly impact the execution time of the entire job-flow.

6.3.3 Storage Issues

In a traditional supercomputing environment, each node is normally attached with only one HDD. This configuration puts a practical limitation on the total number of disk I/O operations per second (IOPS). On the other hand, the big data applications that consider data locality, typically involve a huge volume of data read/write from/to the Direct-Attached-Storage (DAS) of the compute nodes. Therefore, the applications might suffer from I/O wait. Although some variations of Hadoop (e.g., [90]) are optimized to read/write large volume of data from/to

other parallel file systems (e.g., Lustre and GPFS), thus taking advantage of huge amount of IOPS available through the dedicated I/O servers, the performance can be severely constrained by the network bottleneck. Additionally, it will incur extra cost to the cluster. For simplicity, in this work, the HDFS is used as the distributed file system and use the local file system for the shuffled data.

Hadoop involves a huge amount of disk I/O in the entire job flow. For example, at the beginning (and the end) of a Hadoop job, all the mappers read (and the reducers write) a huge volume of data in parallel from (to) the HDFS which is mounted on the DAS device(s) of the compute nodes. Again, in the shuffle phase, a huge volume of intermediate key-value pairs is written by the mappers and subsequently read by the reducers to/from the local file system which is again mounted on the same DAS. Giraph, on the other hand, is an in-memory framework. It reads/writes a huge volume of data from/to the HDFS only during the initial input and the final output.

6.3.4 Memory Issues

The traditional supercomputers normally use a 2GB/core memory as a standard configuration. This causes a significant trade-off between the number of concurrently running workers (mappers or reducers), and the memory used by each of them. Lower memory per worker (lower java heap space) can significantly increase the garbage collection frequency of each worker. Also, in case of Hadoop, smaller memory per worker puts a practical limitation on its buffer size resulting in a huge amount of data spilling to the disk in the shuffle phase, thereby making the job severely I/O-bound especially in case of HDD. Furthermore, the lower memory per node hinders the caching especially for a memory-intensive graph analysis job with Giraph that loads a huge amount of data in memory for iterative computation.

6.4 Evaluation Methodology

6.4.1 Experimental Testbeds

Table-6.1 shows the overview of the experimental testbeds. SuperMikeII, the LSU HPC-cluster, offers a total of 440 computing nodes (running Red Hat Enterprise Linux 6). However, a maximum of 128 can be allocated at a time to a single user. SwatIII is a regular datacenter with 128 compute nodes (running on Ubuntu 12.0.4 LTS). However, a maximum of 16 nodes has been used. SwatIII has been configured in seven different ways to study the pros and cons of different hardware components individually and from the viewpoint of their overall organization in a scaled out and a scaled up environment. For the sake of convenience, each configuration is given a meaningful name as shown in Table-6.1. CeresII is a novel hyperscale system based on Samsung MicroBrick. This study evaluated it as a next-generation cluster which is found to resolve many of the problems in the existing HPC-cluster and the regular datacenter. It is to be noted that a homogeneous configuration has been used across any cluster. The thesis reported the performance and the price of different clusters in terms of the Hadoop datanodes (DN) only. For the masternode, A minimal configuration is used based upon the resources in the cluster. The subsequent sections uses the term node and datanode interchangeably.

Table-6.2 shows the hardware specification used in different clusters and their cost³. The cost of each node of each cluster configuration is calculated (shown in Table-6.1) based upon this. The hardware configuration of SuperMikeII serves as the baseline and compare all the performance results of SwatIII and CeresII, to this baseline. Each node of SuperMikeII and any SwatIII variants has the same number of processors and cores, in particular, 2 8-core Intel SandyBridge Xeon

³Price information is collected from <http://www.newegg.com> and <http://www.amazon.com>. The minimum listed price is considered as per Jun 17, 2015.

Table 6.1: Experimental testbeds with different configurations

	Super MikeII	SwatIII- Basic-HDD	SwatIII- Basic-SSD	SwatIII- Memory	SwatIII-Full Scaleup-HDD/ SSD	SwatIII- Medium-HDD/ SSD	CeresII
Cluster category	HPC-cluster	Scaled out	Scaled out	Memory optimized	Scaled up	Medium-sized	Hyperscale
#Physical-Cores/node	16	16	16	16	16	16	2
DRAM (GB)/node	32	32	32	256	256	64	16
#Disks/node	1-HDD	1-HDD	1-SSD	1-SSD	7-HDD/SSD	2-HDD/SSD	1-SSD
Network	40-Gbps InfiniBand	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Virtual Ethernet
Cost/node (\$)	3804	4007	4300	6526	SSD:9226, HDD:7175	SSD:5068, HDD:4482	879
Blocking	2:1	-	-	-	-	-	-
#Nodes for bubble (90GB)	15	15	15	15	4	2	31
#Nodes for human genome (452GB)	127	-	-	-	15	-	-

Table 6.2: Hardware components of different cluster configurations, and their cost

Hardware component	Used in	Cost (\$)
Intel SandyBridge Xeon 64bit Ep series (8-cores) processor	SuperMikeII, SwatIII	1766
Intel Xeon E3-1220L V2 (2-cores) processor	CeresII	384
Western Digital RE4 HDD	SuperMikeII	132
Western Digital VelociRaptor HDD, 500GB	SwatIII HDD-variants	157
Samsung 840Pro Series SATAIII SSD, 500GB	SwatIII SSD-variants	450
Samsung 840Pro Series SATAIII SSD, 250GB	CeresII	258
Samsung DDR3 16GB memory module	SwatIII, CeresII	159
32GB 1600MHz RAM (decided by Dell)	SuperMikeII	140 (Average)

64bit Ep series processors. To do a fair comparison, the HyperThreading has been disabled in the SwatIII as SuperMikeII does not have it.

The first three variants of SwatIII, SwatIII-Basic-HDD, SwatIII-Basic-SSD, and SwatIII-Memory, are used to evaluate the impact of each individual component of a compute cluster i.e., network, storage and the memory. SwatIII-Basic-HDD is similar in every aspect to SuperMikeII except it uses 10-Gbps Ethernet instead of 40-Gbps InfiniBand as in SuperMikeII. SwatIII-Basic-SSD, as the name suggests, is storage optimized and uses one SSD per node instead of one HDD as in SuperMikeII and SwatIII-Basic-HDD. On the other hand, SwatIII-Memory is both memory and storage optimized, i.e., it uses 1-SSD as well as 256GB memory per node instead of 32GB as in the previous three clusters.

Unlike SuperMikeII or SwatIII-Basic and -Memory which use only one DAS device per node, SwatIII-FullScaleup-HDD/SSD and SwatIII-Medium-HDD/SSD use more than one DAS device (Either HDD or SSD as the names suggest) per node. They also vary in terms of total amount of memory per node. However, the

total amount of storage and memory space is almost same across all these clusters. These clusters have been used to mainly evaluate different types of hardware organizations and architectural balances from the viewpoint of scaled out and scaled up configurations. It is to be noted in case of SwatIII, the chapter uses the term scaled up and out in terms of the amount of memory and number of disks. The number of cores per node is always same. In either of SwatIII-FullScaleup and SwatIII-Medium, JBOD (Just a Bunch Of Disks) configuration is used as per the general recommendation by [2], Cloudera, Hortonworks, Yahoo, etc. Use of the JBOD configuration eliminates the limitation on disk I/O speed, which is constrained by the speed of the slowest disk in case of a RAID (Redundant Array of Independent Disk) configuration. As mentioned in [2], JBOD is found to perform 30% better than RAID-0 in case of HDFS write throughput.

The last one, CeresII, is a novel scaled out architecture based on Samsung MicroBrick. It is an improvement over CeresI [91]. In this chapter, a prototype version of the CeresII cluster is used for the evaluation and study. The next chapter describes the commercial version of the cluster. One MicroBrick chassis of CeresII has 22 computation-modules (or, compute servers). Each module consists of one intel Xeon E3-1220L V2 processor with two physical cores, 16GB DRAM module (Samsung), and one SATA-SSD (Samsung). Each module has several PCI-express (PCIe) ports. Unlike SuperMikeII (traditional supercomputer) and SwatIII (regular datacenter), all the compute servers of CeresII in a single chassis are connected to a common PCIe switch to communicate with each other. The highly dense servers per chassis in CeresII have a total 44 physical cores connected through PCIe comparing to 16 physical cores per node as in SuperMikeII and SwatIII. Furthermore, the use of SSD reduces the I/O wait and 8GB RAM per physical core improves the access parallelism.

6.4.2 Understanding the Workload: Genome Assembly with Hadoop and Giraph

De novo genome assembly problem can be interpreted as a simplified de Bruijn graph traversal problem. The de novo assembly has been classified in two stages as follows: *a)* Hadoop-based de Bruijn graph construction and *b)* Giraph-based graph simplification. Following is a brief overview of the assembler.

1) Hadoop-based De Bruijn graph-construction (data- and compute-intensive workload) The user-defined map function reads each line of the data file in fastq format [92] and filters the lines containing only the nucleotide characters (A, T, G, and C). These lines are known as short reads, which represent a very small fragment of the entire genome. The map task then divides each of those reads into several substrings of length k . These substrings are known as k -mers. Two adjacent k -mers represent a vertex and an edge emitted from that vertex in the de Bruijn graph. The vertex is the key and the edge from it is considered as its corresponding value. Then reduce function collects all the edges emitted from each vertex, aggregate and then writes the graph structure into the HDFS in adjacency-list format. The job may produce terabytes of temporary or shuffled data based on the value of k . For example, for a read-length of 100 and k of 31 the shuffled data size is found to be 20-times than the original fastq input. On the other hand, based upon the number of unique k -mers, the final output (i.e., the graph) can vary from 1 to 10 times of the size of the input [1]

2) Giraph-based Graph Simplification (memory- and compute-intensive workload) This stage consists of a series of memory-intensive Giraph jobs. Each Giraph job consists of three different types of computation: compress linear chains of vertices followed by removing the tip-structure and then the bubble-structure (introduced due to sequencing errors) in the graph. The program maintains a counter on the number of supersteps and the master-vertex class invokes different

computation based on that. The software tool compresses the linear chains into a single vertex using a randomized parallel algorithm implemented with Giraph. The computation proceeds in rounds of two supersteps until a user-defined *limit* is reached. In one superstep, each compressible vertex with only one incoming and outgoing edge is labeled with either *head* or *tail* randomly with equal probability and send a message containing the tag to the immediate predecessor. In the next superstep, all the *head-tail* links are merged, that is, the *head-kmer* is extended (or, appended) with the last character of the *tail-kmer* and the *tail* vertex is removed. Each vertex also maintains a frequency counter which increments after each extension. Tip removal is a two-step process. The first superstep identifies all the vertices with very short length (less than a threshold) and no outgoing edge as tips which are removed in the second superstep. Finally, the bubbles-structures are resolved in another two supersteps. The first super step identifies the vertices with same predecessor and successor as bubble and sends a message to their predecessor with their id, value, and frequency to their corresponding predecessors. The predecessor employs a Levenshtein-like edit distance algorithm. If the vertices are found similar, then the lower frequency vertex is removed.

6.4.3 Input Data

Table-6.3 and 6.4 show the details of the data size used in the assembly pipeline. In this work, two real genome datasets produced by Illumina Genome AnalyzerII, a high throughput next generation sequencing machine. They are as follows: 1) a moderate size bumble bee genome data (90GB) and 2) a large size human genome data (452GB). The corresponding graph sizes are 95GB and 3.2TB (using $k = 31$ in both the cases). The bumble bee genome is available in GAGE [93] website⁴. The Human genome is available in NCBI website with accession number SRX016231⁵.

⁴Genome Assembly Gold-standard Evaluation (<http://gage.cbcb.umd.edu/>)

⁵[http://www.ncbi.nlm.nih.gov/sra/SRX016231\[accn\]](http://www.ncbi.nlm.nih.gov/sra/SRX016231[accn])

Table 6.3: Moderate-size bumble bee genome assembly

	Job Type	Input	Final output	# jobs	Shuffled data	HDFS Data
Graph Construction	Hadoop	90GB (500-million reads)	95GB	2	2TB	136GB
Graph Simplification	Series of Giraph jobs	95GB (71581898 vertices)	640MB (62158 vertices)	15	-	966GB

Table 6.4: Large-size human genome assembly

	Job Type	Input	Final output	# jobs	Shuffled data	HDFS Data
Graph Construction	Hadoop	452GB (2-billion reads)	3TB	2	9.9TB	3.2TB
Graph Simplification	Series of Giraph jobs	3.2TB (1.5-billion vertices)	3.8GB (3-million vertices)	15	-	4.1TB

6.4.4 Hadoop Configurations and Optimizations

Since the goal is to evaluate the underlying hardware components and their organizations, any unnecessary change in the source code of Hadoop or Giraph has been avoided. Cloudera-Hadoop-2.3.0 and Giraph-1.1.0 have been used for the entire study and use the Cloudera-Manager-5.0.0 for monitoring the system behavior. This section provides the evaluation methodology in details. It is worthy to mention here, although the benchmark genome assembler (GiGA) has been used for the evaluation purpose, the systematic analysis can be easily applied to other data-intensive applications without any modification. To evaluate the relative merits of different clusters, the evaluation started with tuning and optimizing different

Hadoop parameters to the baseline, that is a traditional HPC cluster, SuperMikeII. Then, the parameters have been further modified with the change in the underlying hardware infrastructure in SwatIII cluster to optimize the performance in each configuration. A brief description of the Hadoop-parameters is as follows.

1) Number of concurrent YARN containers: After performing rigorous testing, it is observed, 1-HDD/DN has a practical limitation on this number. For SuperMikeII and SwatIII-Basic-HDD (1-HDD/DN cases), 8-containers/DN (i.e., half of total cores/node) produce the best result. For any other cluster, the number of concurrent containers per datanode is kept equal to the number of cores per node.

2) Amount of memory per container and Java-heap-space: In each node of any cluster, 10% of the DRAM is kept for the system's use. The rest of the DRAM is equally divided among the concurrently launched containers. The Java heap space per worker is always kept lower than DRAM per container as per Hadoop's recommendation.

3) Total number of Reducers: After observing the job profiles over multiple datasets, we concluded that 2-times of reducers than the number of concurrent containers gives the best performance.

4) Giraph workers: The number of Giraph workers is set according to the number of concurrent YARN containers.

5) Other Giraph parameters: Enough memory is used always to accommodate the graph structure in memory and always avoided using the out-of-core execution feature of Giraph, which writes huge data to the disk.

6.5 Impact of Different Hardware Component

This section compares the individual impact of each hardware component, such as, network, storage, and memory individually on the benchmark genome assembler. To do that, 16 nodes in both SuperMikeII and SwatIII has been used. Each node in both the clusters has 16 processing cores. We started by comparing

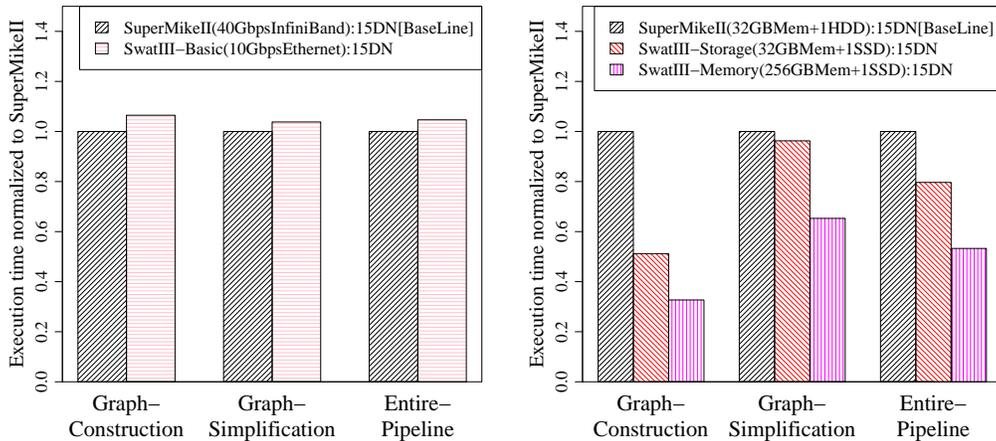
the impact of the network between SuperMikeII and SwatIII-Basic-HDD. Then, we further optimized those 16 nodes of SwatIII cluster incrementally in terms of storage by providing SSD (named as SwatIII-Basic-SSD) and then providing more memory to each node (named as SwatIII-Memory). The reported execution-times are the means of at least 3 runs of the assembler on each of the cluster configurations.

6.5.1 Effect of Network: InfiniBand vs Ethernet

Figure-6.1a compares the impact of network interconnect on each stage of PGA's genome assembly pipeline while assembling a 90GB bumble bee genome. The execution time is normalized to the SuperMikeII-baseline. No visible performance difference (less than 2%) has been noticed on any of the stages of the assembly pipeline even though SuperMikeII uses 40-Gbps QDR InfiniBand whereas SwatIII-Basic-HDD uses a 10-Gbps Ethernet. The reason is as follows: although the average latency in SuperMikeII is almost 1/14 of that in SwatIII (0.014ms in SuperMikeII compare to 0.2ms in SwatIII), the average effective bandwidth between any two compute nodes of SuperMikeII was found to be almost 10-times lower than that of SwatIII (954Mbit/s in SuperMikeII, whereas 9.2Gbit/s in SwatIII) because of the 2 : 1 blocking ratio in the InfiniBand network.

6.5.2 Effect of Local Storage Device: HDD vs SSD

Figure-6.1b compares the execution time of SwatIII-Basic-SSD (1-SSD/node) to the SuperMikeII-baseline (1-HDD/node). The second column of each stage of the assembler in Figure-6.1b shows the impact of using SSD in that stage of the assembly. It is observed that almost 50% improvement in the shuffle intensive graph-construction stage because of reduced I/O wait. However, graph-simplification, a series of in-memory Giraph jobs (that read/write data only to the HDFS), is not affected much (less than 3%) by using SSD.

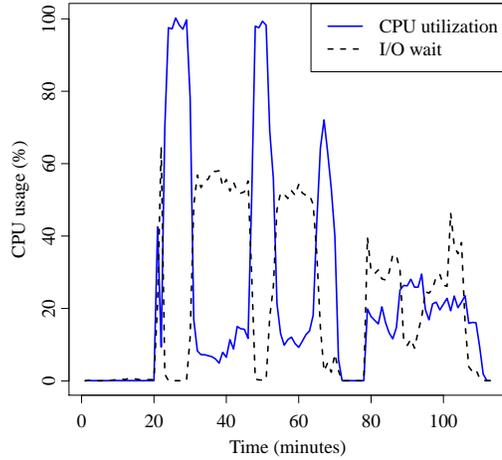


(a) Effect of network (InfiniBand vs Ethernet). (b) Effect of local storage (HDD vs SSD) and size of DRAM.

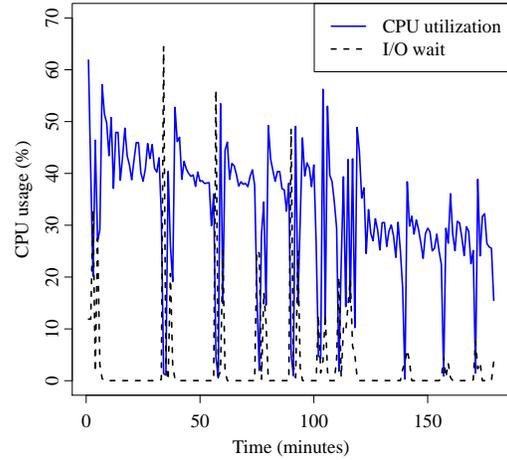
Figure 6.1: Impact of each individual hardware component on execution time of the assembly pipeline in 15-DN

Actually, the shuffle phase of the Hadoop job experiences a lot of I/O wait when a large number of I/O threads work concurrently to spill a huge amount of data to the disk (by the mappers) and subsequently read by the reducers. Giraph shows I/O wait only when it reads/writes a large graph from/to HDFS (Figure-6.2b). I/O wait is significantly reduced using SSD (Figure-6.2c and -6.2d) instead of HDD which improves the Hadoop performance remarkably. However, for Giraph no notable performance improvement has been observed using SSD because of very less I/O wait.

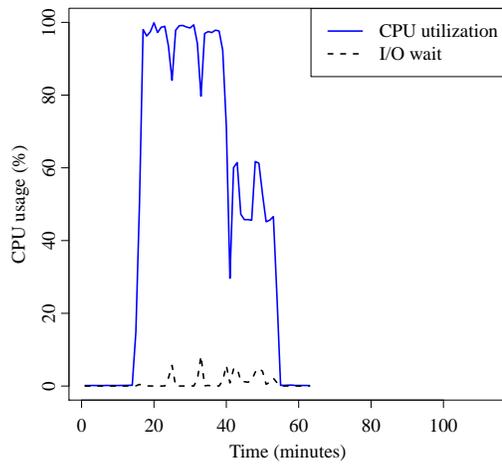
The system behavior is shown in Figure 6.2 and 6.3. Basically, an SSD increases the disk IOPS per DataNode by 7 to 8 times than an HDD especially during the shuffle phase of Hadoop which writes a vast amount of data to the local file system as shown in Figure-6.3a. In case of Giraph, the corresponding improvement is 1.5-times as shown in Figure-6.3b. Considering the I/O throughput to HDFS, it is also observed that also observed 1.5-times improvement in case of SSD for both Hadoop and Giraph as shown in Figure-6.3c and -6.3d. Giraph, which writes data



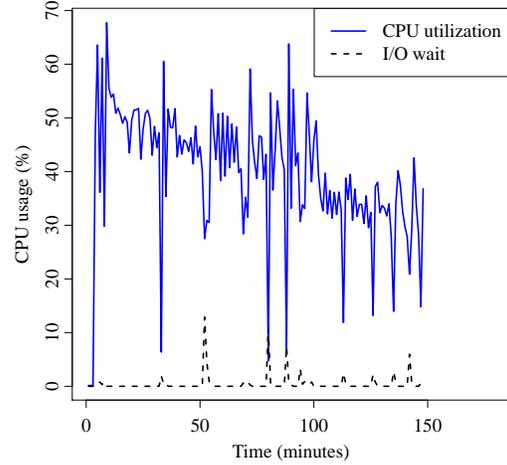
(a) Hadoop-based graph-construction in SwatIII-Basic-HDD (1-HDD/node)



(b) Giraph-based graph-simplification in SwatIII-Basic-HDD (1-HDD/node)

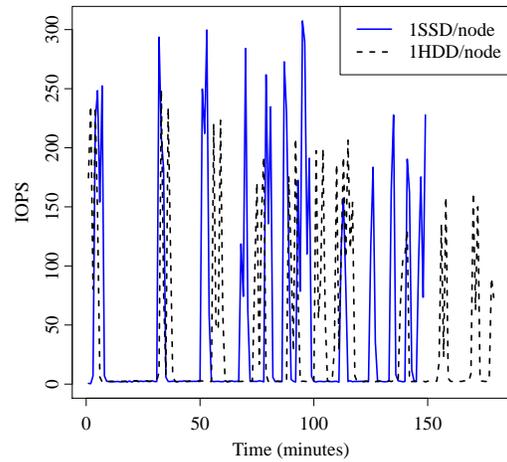
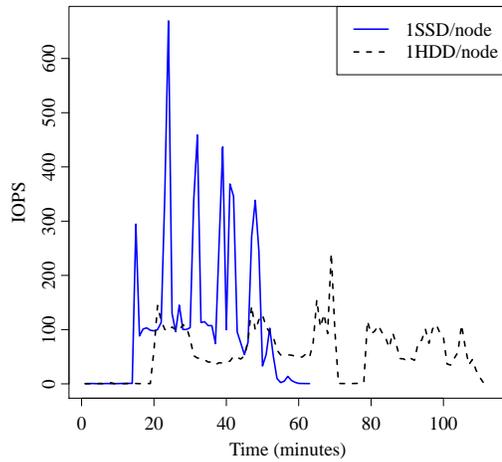


(c) Hadoop-based graph-construction in SwatIII-Basic-SSD (1-SSD/node)

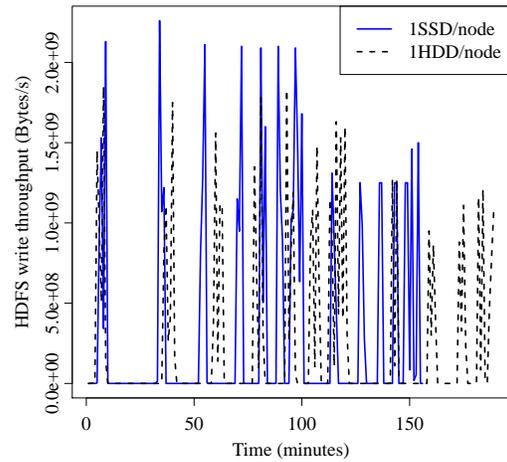
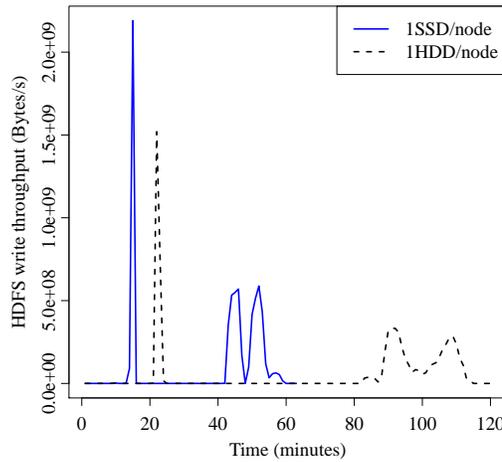


(d) Giraph-based graph-simplification in SwatIII-Basic-SSD (1-SSD/node)

Figure 6.2: CPU utilization and I/O wait characteristics in SwatIII-Basic-HDD (1-HDD/node) and SwatIII-Basic-SSD (1-SSD/node)



(a) WriteIOPS/DN in graph-construction in SwatIII-Basic-HDD (1-HDD/node) and in SwatIII-Basic-SSD (1-SSD/node) (b) WriteIOPS/DN in graph-simplification in SwatIII-Basic-HDD (1-HDD/node) and SwatIII-Basic-SSD (1-SSD/node)



(c) HDFS-Write-throughput in graph-construction in SwatIII-Basic-HDD (1-HDD/node) and SwatIII-Basic-SSD (1-SSD/node) (d) HDFS-Write-throughput in graph-simplification in SwatIII-Basic-HDD (1-HDD/node) and SwatIII-Basic-SSD (1-SSD/node)

Figure 6.3: Comparison of disk-write IOPS (write) on local file system (of each datanode) and I/O throughput for HDFS-write (across all datanodes) for HDD and SSD

to the HDFS only, shows the similar I/O characteristics for both IOPS per DataNode (DN) and HDFS I/O throughput because they are related by the equation: $HDFS_IO_Throughput = IOPS_per_DN \times Bytes_per_IO \times \#DN$, where $Bytes_per_IO$ is the characteristics of the disk. However, the HDFS throughput characteristics across all the DN varies significantly from the IOPS per DN in case of a shuffle-intensive Hadoop job, which writes lots of data to the local file system.

6.5.3 Effect of Size of DRAM

The third columns of Figure-6.1b shows the impact of increasing the amount of memory per node. It is observed that almost 20% improvement in the initial graph-construction phase from SwatIII-Basic-SSD, i.e., almost 70% improvement to the baseline. In the Giraph phase, the corresponding improvement is 35%. The improvement is because of the caching especially in case of Giraph, where computation proceeds in iterative supersteps. A huge amount of data is kept in cache and is fetched upon requirement during the next compute-superstep.

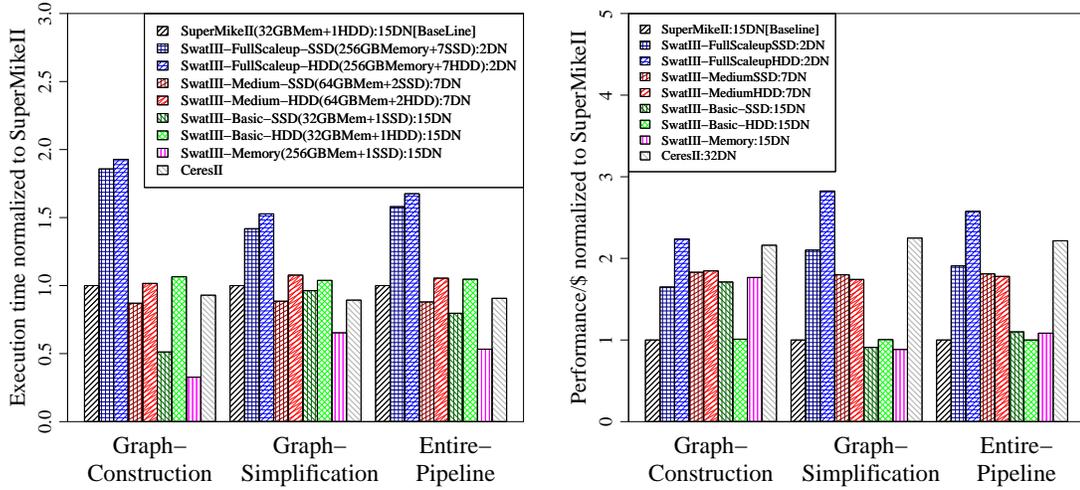
6.6 Impact of Different Hardware-Organizations

This section compares different cluster architecture in terms of raw execution time as well as performance-to-price. Again, the execution times are the averages of at least 3 runs of the assembler on each different hardware configuration.

6.6.1 Execution Time Comparison between SuperMikeII and SwatIII Variants (with Moderate Size Bumble Bee Genome)

Figure-6.4a shows the relative merits of different cluster architectures in terms of execution time. The total aggregated storage and memory space are kept almost same across all the clusters (Except the SwatIII-Memory). The assumption behind this experimental setup is that the total amount of data should be held entirely in any of the clusters that cannot be compromised. The observations are as follows:

1. SwatIII-Basic: As discussed earlier in Section-6.5, for Hadoop, the SSD variant of this scaled out cluster (32GB-RAM + 1-disk/DN & 16-DNs) shows



(a) Execution time (Lower execution time means better performance) (b) Performance-to-Price (Higher performance per dollar is better)

Figure 6.4: Performance comparison among different type of cluster architectures in terms of normalized execution time and performance-to-price

2x speedup over the baseline whereas the HDD variant performs similarly to the baseline. For Giraph, both of them perform almost similar to the baseline.

2. SwatIII-FullScaleup: This scaled up (256GB-RAM + 7-disks/DN) small sized cluster (only 2-DNs) takes the maximum time for any workload because of least number of processing cores among all. Observe that for Hadoop, both SSD and HDD variants of this scaled up cluster perform similarly, which is in contrast with scaled out cluster (SwatIII-Basic). Section-6.6.4 discusses it in more detail.

3. SwatIII-Medium: The HDD variant of this cluster (64GB-RAM + 2-disks/DN & 7-DNs) performs almost similar to the baseline for both Hadoop and Giraph even though the total number of cores in the cluster is half of the baseline. This is because of 2-HDDs and 64GB RAM per node increase the IOPS and the caching respectively. The SSD variant performs slightly better than the HDD because of further increase in IOPS.

4. SwatIII-Memory: The performance characteristics of this cluster is discussed earlier in Section-6.5. It is no surprise that this configuration (256GB-RAM

+ 1-SSD/DN & 16DNs) shows the lowest execution time among all because of the huge amount of memory available across the cluster.

6.6.2 Performance-to-Price Comparison between SuperMikeII and SwatIII Variants (with Moderate Size Bumble Bee genome)

It is considered that the performance as the inverse of the execution time and divided it by the total cost of the cluster to get the performance/\$. Since all the clusters have the same amount of storage and memory space (except SwatIII-Memory), none of them get any price benefit over another because of the total storage or memory space. Rather, the performance to price has been compared from the viewpoint of a proper architectural balance among the number of cores, number of disks, and amount of memory per node. The chapter did not consider the cost of the network for a fair comparison with SuperMikeII, the public HPC-cluster that is shared among many users.

Figure-6.4b compares the performance/\$ metric among all the clusters. The observations are as follows:

1) SwatIII-Basic: For Hadoop, the SSD variant of this scaled out cluster shows 2-times better performance/\$ comparing to the baseline as well as to its HDD variant. However, for Giraph, it does not add any benefit. The HDD variant, as expected, shows a similar result as the baseline.

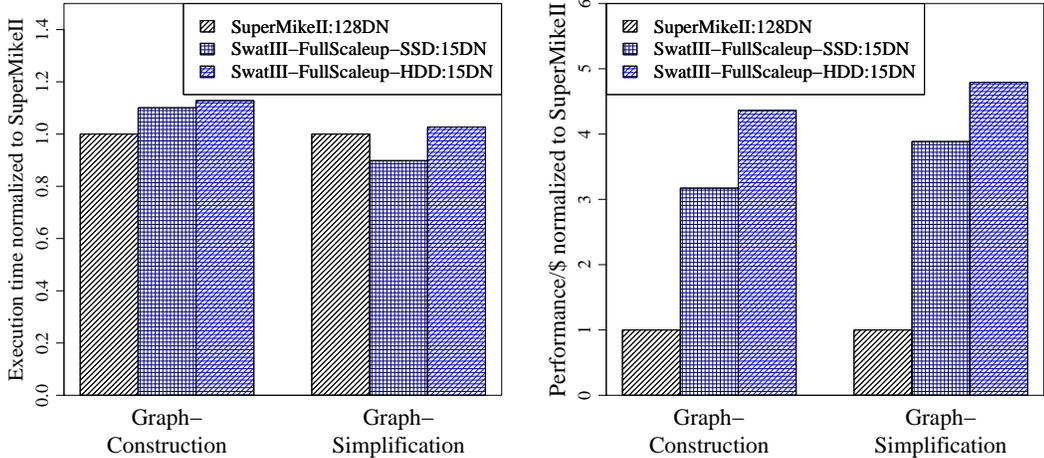
2) SwatIII-FullScaleup: Although the performance of this scaled up cluster is the lowest in terms of execution time, it shows high performance/\$ for both Hadoop and Giraph. For Hadoop, the SSD and HDD variants show 1.5 and 2.5 times benefit to the baseline respectively. For Giraph, the corresponding benefit is 2 and 3 times respectively for SSD and HDD. Due to the similar execution time in both HDD and SSD variant of this scaled up cluster, the HDD variant obviously shows better performance/\$ than the SSD variant. This is again in contrast with the scaled out (SwatIII-Basic) case

3) SwatIII-Medium: Both the HDD and SSD variant of this configuration shows a similar result, almost 2-times better than the baseline for both Hadoop and Giraph. Considering both performance and the price, it is the most optimal configuration in the evaluation.

4) SwatIII-Memory: For Hadoop, it shows 2-times benefit to the baseline. However, once SSD is used as the underlying storage (comparing to SwatIII-Basic-SSD) more memory does not add any advantage in terms of performance/\$. For Giraph, it does not have any impact on performance/\$ compared to the baseline.

6.6.3 Comparing SuperMikeII and SwatIII (with large human-genome)

The large human genome (452GB) produces huge amount of shuffled data (9.9TB) as well as the graph data (3.2TB). This work used 127 DataNodes in SuperMikeII (32GB-RAM + 1-disk/DN) and 15 DataNodes in the SwatIII-Full-Scaleup (256GB-RAM + 7-disks/DN) HDD and SSD. Figure-6.5a and 6.5b shows the execution time and the performance/\$ respectively for the human genome assembly pipeline. The observations are as follows:



(a) Execution time (Lower execution time is better). (b) Performance to price (Higher performance/\$ is better).

Figure 6.5: Comparison of different types of cluster architecture for human genome assembly pipeline.

1) For Hadoop, the 127-DNs of SuperMikeII (2032-cores) show only 15-17% better performance than 15-DNs (240 cores) of SwatIII-FullScaleup cluster (any variant) while using almost 9-times more cores. The reasons behind the lower performance in SuperMikeII are both the huge I/O and network bottleneck as discussed earlier in Section-6.3 and -6.5. Again, observe that for this large dataset also, both the SSD and HDD variants of SwatIII-FullScaleup perform similarly as observed in Section-6.6.1.

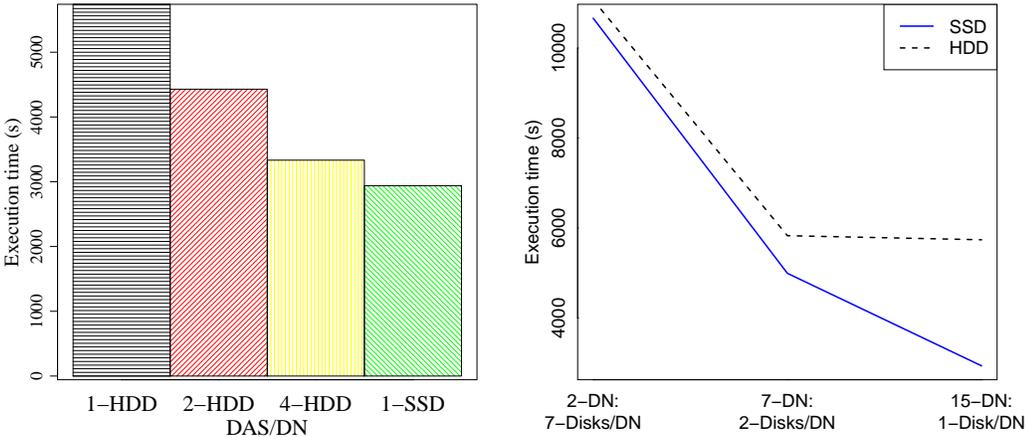
2) For the Giraph-based graph-simplification stage also, SuperMikeII did not show the expected performance mainly because of the network bottleneck. To analyze the terabyte scale graph, Giraph passes a huge amount of messages across the workers, which experience significant bottleneck by the 2:1 blocking ratio in the InfiniBand network and the lower effective bandwidth between compute nodes of SuperMikeII.

3) In terms of performance per dollar, the scaled up configuration shows huge benefit over the baseline. For Hadoop, the gain is 3 to 5 times based on the storage media. For Giraph, the corresponding gain is almost 4 to 5 times. Again, because of the similar execution time, the HDD variant of SwatIII-FullScaleup shows better performance per dollar than the SSD variant.

6.6.4 Performance of SSD in Scaled Out and Scaled Up Cluster

Storage optimized, scaled up cloud instances frequently come with 4 to 8-SSDs per instance to improve the performance, consequently incurring high setup-cost as well as service-charge. For example, AWS-i2.8xlarge offers 8-SSDs per instance at a rate of \$6.82/hour, which is one of the high-cost AWS-EC2-instances. But, is it the effective way to deploy the SSDs? The disk controllers saturate after a certain threshold (an observation by Szalay [94]). In this section, we compare the performance characteristics of HDD and SSD from the perspective of scaled out and scaled up configuration.

Figure-6.6a compares the performance of a single SSD and increasing number of HDDs per node for the Hadoop-based graph-construction stage of the bumblebee genome assembly pipeline while using a total of 15-DNs. The performance improves almost linearly by increasing the number of HDDs per DataNode in the cluster. On the other hand, 4-HDDs per node shows similar performance (only 5% variation) with a single-SSD per node. At this point the job is CPU-bound, and adding more disk(s) to the DataNodes does not improve the performance. Consequently, any significant performance improvement is not expected after this point (that is, more than 4-disks per node). Both Figure-6.6b as well as Figure-6.5a substantiate the claim for moderate-size bumble bee and large-size human genome data. It can be clearly observed that for both the data, both the HDD and SSD variant of SwatIII-FullScaleup perform similarly where each DataNode is equipped with 7-disks. However, the SSD showed a significantly better performance than HDD when scaled out by adding more compute nodes to the cluster because a lower number of HDDs make the job I/O bound as shown in Figure-6.6b.



(a) Hadoop performance trend using 1, 2 (b) Hadoop performance trend for SSD and 4 HDD(s) and 1-SSD per node using and HDD using 1, 2, 7 disks per node in 15 datanodes in the cluster. 15, 7 and 2 datanodes in the cluster.

Figure 6.6: Performance trend using HDD and SSD in Hadoop. SSD shows better performance and scalability in a scaled out environment

6.6.5 Performance of CeresII prototype: Samsung MicroBrick with PCIe Communication

In this section, The chapter evaluates a Samsung MicroBrick-based novel architecture, called CeresII. As mentioned before, CeresII uses 2 physical cores, 1-SSD and 16GB memory per compute server and uses a PCIe-based interface to communicate among high-density compute-servers in a chassis. For communication between different chassis, it uses 10-Gbps Ethernet. The PCIe based communication enables building highly efficient distributed clusters of extremely low communication overhead. At the same time, the SSD based MicroBricks enables highly efficient resource utilization at a significantly lower cost.

To assemble the 95GB bumble bee genome 32 compute-servers of CeresII were used as Hadoop datanodes. The last columns in Figure-6.4a and Figure-6.4b show the execution-time and performance/\$ respectively for CeresII for different stages of the assembly. CeresII shows the similar execution time to the baseline in every stage of the assembly pipeline while giving almost 2-times improvement in performance/\$.

From the performance comparison between SuperMikeII and SwatIII, a huge trade-off between the execution time and the performance/\$ was noticed. For example, even though the full scaled up small-sized clusters (2-DNs cases) show extremely low performance, they show a magnitude higher performance/\$. It is also concluded that the medium-sized clusters (7-DNs) are well balanced considering both performance and cost. On the other hand, CeresII shows similar execution time as the medium sized clusters (which is eventually same as the baseline) and better performance per dollar. Moreover, the Samsung MicroBrick based architecture consumes less power and space [91]. Hence, it is concluded that CeresII shows the maximum benefit in terms of TCO (total cost of ownership).

6.7 Conclusion

This work analyzed the performance characteristics of two popular state of the art big data analytic software tools, Hadoop and Giraph, on top of different distributed-cyber-infrastructures with respect to a real-world data- and compute-intensive HPC workload. The research pointed out several limitations in a traditional HPC cluster, both, in individual node layer (e.g., memory and storage) as well as network interconnect layer. The novel MicroBrick-based CeresII-cluster with low-power but high-density compute nodes connected with PCIe-based communication interface is a good future direction to alleviate many of the existing architectural limitations.

The research also pointed out the huge trade-off between the performance and the price that the data- and memory-intensive HPC applications experience with the traditional deployment of the existing hardware components. The existing distributed-cyber-infrastructures should be modified significantly in order to provide good performance while staying within the budget. It is indeed the future direction of the work. CeresII, from that perspective, also provides a very good initial starting point.

Chapter 7

A Theoretical Model for Cost-Balanced HPC Cluster for Data Science

High-performance analysis of big data demands more computing resources, forcing similar growth in computation cost. So, the challenge to the HPC system designers is providing not only high performance but also a high performance at lower cost. For high performance yet cost effective cyberinfrastructure, the thesis proposes a new system model augmenting Amdahls second law for a balanced system to optimize price-performance-ratio. The optimal balance among CPU-speed, I/O-bandwidth and DRAM-size (i.e., Amdahls I/O- and memory-number) are expressed in terms of application characteristics and hardware cost. Considering Xeon processor and recent hardware prices, the analysis showed that a system needs almost 0.17GBPS I/O-bandwidth and 3GB DRAM per GHz CPU-speed to minimize the price-performance-ratio for data- and compute-intensive applications.

The thesis substantiates the claim evaluating three different cluster architectures: 1) SupermikeII, a traditional HPC cluster, 2) SwatIII, a regular datacenter, and 3) CeresII, a MicroBrick-based hyperscale system. CeresII with 6-Xeon cores (2GHz/core), 1-NVMe SSD (2GBPS I/O-bandwidth) and 64GB DRAM per node, closely resembles the optimum produced by the model. Consequently, CeresII shows better price-performance-ratio than both SupermikeII (65-85%) and SwatIII (40-50%) for data- and compute-intensive Hadoop benchmarks (TeraSort and WordCount) and a genome assembler developed using Hadoop and Giraph.

7.1 Introduction

As the scientific research is becoming more data-driven in nature, it is obvious that providing more resources to an HPC cluster (processing speed, I/O bandwidth,

This chapter previously appeared in IEEE Cloud 2017 [95]. Reprinted by permission.

DRAM) will improve the performance of data-intensive scientific applications. But at what cost? So, the major challenge to the system designers nowadays is not in providing only high performance but in providing expected performance in reduced cost (i.e., minimizing price-performance-ratio).

At this inflection point of HPC landscape, system designers must consider more degrees of freedom for new cluster architecture for big data processing than for existing HPC clusters which focus only on doing the calculation at blazing speed. They must address such questions as to how much I/O bandwidth is required per processing core? How much memory is required to optimize performance and cost? These complex performance and economic factors together motivate new designs of HPC infrastructure.

With this motivation, this work makes an initial attempt to resolve the existing performance and cost conundrum by augmenting Amdahl's second law (i.e., Amdahl's I/O and memory number) for balanced system. The thesis proposes a simple additive model to optimize price-performance-ratio by quantifying the system balance between CPU speed, I/O bandwidth, and size of DRAM in terms of software application characteristics and the current trend in hardware cost. The final outcome of this model are the two modified Amdahl's numbers which can be easily used by hardware vendors to propose a cost-effective architecture for data- and compute-intensive applications.

Assuming an equal distribution of I/O and compute work in a data-intensive application, the model suggests that a balanced HPC system needs almost 0.17-GBPS I/O bandwidth, and almost 3-GB of DRAM per GHz of CPU speed using Intel Xeon processor and current price trend of different hardware.

To substantiate the claim, three different cluster architectures: 1) SupermikeII, a traditional HPC cluster, 2) SwatIII, a regular datacenter, and 3) CeresII, a MicroBrick-based novel hyperscale system are evaluated. CeresII with 6-

Xeon-D1541 cores (2GHz/core), 1-NVMe SSD (2GBPS I/O-bandwidth) and 64GB DRAM per node, closely resembles the optimum produced by the model. Consequently, it outperformed both the clusters for data- and compute-intensive Hadoop benchmarks (TeraSort and WordCount) as well as the benchmark genome assembler developed in the thesis (Chapter 4) based on Hadoop and Giraph. Overall, CeresII showed 65-85% and 40-50% better price-performance-ratio over Super-MikeII and SwatIII respectively.

The rest of the chapter is organized as follows: Section-7.2 describes the prior work related to the current effort. Section-7.3 discusses Amdahl's second law in detail. Section-7.4 describes the proposed model. Then, Section-7.5 shows the details of the experimental testbeds and classify those using the proposed model. Section-7.6 describes the evaluation methodology for these clusters (i.e., the details of the software and benchmarks that are used in the evaluation). Section-7.7 discusses the experimental results. Finally, Section-7.8 concludes the chapter with possible improvement to stimulate discussion and future work.

7.2 Related Work

Numerous studies have been performed evaluating the performance implication of different big data analytic software tools (e.g., Hadoop) on different types of hardware infrastructures.

Hardware evaluation studies such as, [80], [81], [82], [84], etc. unanimously concluded that the use of SSD can accelerate the Hadoop applications with respect to standard benchmark (e.g., TeraSort, WordCount, etc.). On the other hand, [87], [88], and [71] evaluated the overall cluster architecture for different Hadoop benchmark. Although these studies provide good insight on the performance of different hardware, the rapid changes in hardware technologies and the cost limits their scope among hundreds of different architectural alternatives.

Simulation studies such as, SimMR [96], MRSim[97], MRPerf [98], etc. reduced

the hardware cost by creating virtual Hadoop job over simulated hardware environment. Although simulation is more cost-effective than real hardware, it comes with lots of software overhead. Also, the process of finding alternative architecture is mostly driven by a trial-and-error method and prior experiences. Considering the broad range of available hardware alternatives with more than 200 Hadoop parameters, it is challenging to provide optimal hardware configuration and may suffer from reliability issues [99].

Analytical models such as [99], [100], [101], etc. abstract away several performance parameters and predict the performance of a Hadoop job mainly using single- or multi-layer queuing networks. Although no overhead is involved in analytical approach, like simulation it is hard to find an optimally balanced architecture in the vast range of hardware alternatives.

To date, the most practical approach to design a balanced system is to follow Amdahl's second law. Computer scientist Gene Amdahl postulated that a balanced system needs 1bit of sequential I/O per second (Amdahl's I/O number) and 1byte of memory (Amdahl's memory number) per CPU instruction per second. Amdahl's second law (with Gray's amendment) can be used for system characterization and proposing a balanced system. For example, Bell and Gray [102] classified the existing supercomputers based upon Amdahl's second law to clarify the future roadmap of the HPC architecture. Cohen [103] applied Amdahl's second law to the datacenter cluster to study the interplay between processor architecture and network interconnect in a datacenter. Chang [104] used Amdahl's second law to better understand the performance of hardware design implications of data analytic systems. Szalay [94], using Amdahl's second law, proposed a new cluster architecture based on SSD and low power processors (such as, Intel Atom, Zotac etc.) to achieve a balance between performance and energy efficiency.

Unlike these studies, this work considers a balanced system as one that opti-

mizes both cost and performance. Hence, this research did not consider the optimal balance of the system (i.e., the I/O and memory ratio to the processor speed) as constants as in original Amdahl's I/O and memory number. Instead, the thesis proposes an additive model to express the optimal system balance as a function of both application characteristics and hardware price.

7.3 Background

7.3.1 Original Form of Amdahl's Second Law

Computer scientist Gene Amdahl postulated several design principles in late 1960 for a balanced system. As mentioned earlier, these design principals are collectively known as Amdahl's second law, which is as follows:

1) Amdahl's I/O law: A balanced computer system needs one bit of sequential I/O per second per instruction per second. From this point, this law will be mentioned as Amdahl's I/O number. Alternatively, Amdahl's I/O number of a balanced system can be expressed as 0.125 GBPS/GIPS (by changing in conventional units).

2) Amdahl's memory law: A balanced computer system needs one byte of memory per instruction per second. From this point, this law will be mentioned as Amdahl's memory number.

Using the notations in Table 7.1, Amdahl's I/O and memory numbers can be expressed as, $\beta_{io}^{opt} = 0.125$ and $\beta_{mem}^{opt} = 1$.

7.3.2 Gray's Amendment to Amdahl's Second Law

Computer scientist Jim Gray reevaluated and amended Amdahl's second law in the context of modern data engineering. These amendments are collectively known as Gray's law. The revised laws are as follows:

1) Gray's I/O law: A system needs 8 MIPS/MBPS I/O (same as Amdahl's I/O number, but in a different unit), but the instruction rate and I/O rate must

be measured on the relevant workload.

2) Gray's memory law: The MB/MIPS (alternatively, GB/GIPS) ratio is rising from 1 to 4. This trend will likely continue.

The underlying implication of Gray's I/O Law is that it aims for systems whose Amdahl's I/O number matches the Amdahl's I/O numbers of the applications (i.e., application balance) that run on that system. In the memory law, Gray simply put forward the statistics reflecting the contemporary state of the cluster architecture.

Using the notations in Table 7.1 Gray's laws can be expressed as, $\beta_{io}^{opt} = \gamma_{io}$ and $\beta_{mem}^{opt} = 4$

7.3.3 Limitations of Existing Laws

Amdahl's second law for balanced systems does not consider the impact of application balance (or applications' resource requirement). Because of the diverse resource requirements, a one-size-fits-all design as suggested in the original law, cannot satisfy the different resource balance ratios for a collection of analytic applications.

Gray's law is more realistic in the sense that it considers the impact of application balance on the cluster architecture. However, it is limiting to reflect the interplay between application and cost balance. The cost of hardware components has already changed the performance point and will keep on changing it as the technology continues to advance.

7.4 Proposed Model for System Balance

7.4.1 Problem Definition

Using the notations described in Table 7.1, the optimal system balance (i.e., β_{io}^{opt} and β_{mem}^{opt}) needs to be expressed as a function of application balance (i.e., γ_{io} and γ_{mem}) and cost balance (i.e., δ_{io} and δ_{mem}). Mathematically, it can be expressed as, $\beta_{io}^{opt} = f_1(\gamma_{io}, \delta_{io})$ and $\beta_{mem}^{opt} = f_2(\gamma_{mem}, \delta_{mem})$

Table 7.1: Notations used in the model and their meaning

R_{cpu}	CPU speed of a given system S (GHz)
R_{io}	I/O bandwidth of system S (GBPS)
R_{mem}	DRAM size of system S (GB)
W_{cpu}	Fraction of work done by the CPU for a given application W
W_{io}	Fraction of work done by the disk(s) for W
W_{mem}	Fraction of work done by DRAM for W
P_{cpu}	Price per GHz of CPU speed
P_{io}	Price per GBPS of I/O bandwidth
P_{mem}	Price per GB of DRAM
β_{io}	System balance between I/O bandwidth and CPU speed for system S ($= R_{io}/R_{cpu}$)
β_{mem}	System balance between DRAM size and CPU speed for system S ($= R_{mem}/R_{cpu}$)
γ_{io}	Application balance between CPU and I/O bandwidth for application W ($= W_{io}/W_{cpu}$). This term quantifies to what extent the application is I/O- or CPU-intensive. Lower value means more CPU-intensive, higher means I/O-intensive. 1 represents both I/O- and CPU-intensive
γ_{mem}	Application balance between CPU and DRAM size for application W ($= W_{mem}/W_{cpu}$). This term quantifies to what extent the application is memory- or CPU-intensive. Lower value means more CPU-intensive, higher means memory-intensive. 1 represents both memory- and CPU-intensive
δ_{io}	Cost balance between CPU and I/O bandwidth for system S ($= P_{io}/P_{cpu}$)
δ_{mem}	Cost balance between CPU and DRAM for system S ($= P_{mem}/P_{cpu}$)
β_{io}^{opt}	Optimal system balance between I/O bandwidth and CPU speed (The problem under consideration)
β_{mem}^{opt}	Optimal system balance between DRAM size and CPU speed (The problem under consideration)

7.4.2 Model Assumptions

For simplicity of calculation and better usability, the model first ignores the CPU microarchitecture as in [94]. That is, the number of instruction executed per cycle (IPC) is considered as proportional to CPU core frequency. Hence, express the balance between I/O and CPU in terms of GBPS/GHz, and balance between DRAM size and CPU in terms of GB/GHz.

Second, for simplicity, the model is assumed to be additive. That is, the overlap

between work done by I/O, and memory subsystem is ignored. This way, the total execution time (T_{total}) of an application can be written as:

$$T_{total} = T_{cpu} + T_{io} + T_{mem} \quad (7.1)$$

$$\implies T_{total} = \frac{W_{cpu}}{R_{cpu}} + \frac{W_{io}}{R_{io}} + \frac{W_{mem}}{R_{mem}} \quad (7.2)$$

Third, this work assumes the total cost of the system as the summation of individual cost of CPU, I/O, and memory subsystems only. Several constant components such as base cost, service cost, etc. are ignored. This way, the total system cost (C_{total}) can be written as:

$$C_{total} = C_{cpu} + C_{io} + C_{mem} \quad (7.3)$$

$$\implies C_{total} = P_{cpu}R_{cpu} + P_{io}R_{io} + P_{mem}R_{mems} \quad (7.4)$$

7.4.3 Model Derivation

In this section the main objective is to minimize the price-performance-ratio (denoted as f_{cp}). Assuming the performance as the inverse of the execution time, f_{cp} can be expressed as:

$$f_{cp} = C_{total} \times T_{total} \quad (7.5)$$

$$\implies f_{cp} = (C_{cpu} + C_{io} + C_{mem}) \times (T_{cpu} + T_{io} + T_{mem}) \quad (7.6)$$

$$\begin{aligned}
\implies f_{cp} &= C_{cpu}T_{cpu} + C_{cpu}T_{io} + C_{cpu}T_{mem} \\
&\quad + C_{io}T_{cpu} + C_{io}T_{io} + C_{io}T_{mem} \\
&\quad + C_{mem}T_{cpu} + C_{mem}T_{io} + C_{mem}T_{mem}
\end{aligned} \tag{7.7}$$

Assuming a CPU core cannot perform disk and memory operation at the same time, $C_{io}T_{mem}$ (term-6) and $C_{mem}T_{io}$ (term-8) depicts the depreciation of one component when the other is in use. That is, when the memory is in use the cost of disk is depreciated to zero and vice versa. Hence, term-6 and 8 of Equation 7.7 are practically insignificant and should be eliminated from Equation 7.7.

Then, by expanding all the time (T) and cost (C) terms using Equation 7.2 and 7.4 respectively and then substituting with the notation used for system balance in Table 7.1 (i.e., β_{io} and β_{mem}), Equation 7.7 can be rewritten as:

$$\begin{aligned}
f_{cp} &= P_{cpu}W_{cpu} + \frac{1}{\beta_{io}}P_{cpu}W_{io} + \frac{1}{\beta_{mem}}P_{cpu}W_{mem} \\
&\quad + \beta_{io}P_{io}W_{cpu} + P_{io}W_{io} \\
&\quad + \beta_{mem}P_{mem}W_{cpu} + P_{mem}W_{mem}
\end{aligned} \tag{7.8}$$

Again, assuming a CPU core cannot perform disk and memory operation at the same time, partial differentiation with respect to β_{io} and β_{mem} can separately lead us to the optimum system balance in terms of I/O bandwidth and DRAM size respectively, with respect to processing speed.

Partially differentiating with respect to β_{io} , we get:

$$\frac{\partial f_{cp}}{\partial \beta_{io}} = -\frac{1}{\beta_{io}^2}P_{cpu}W_{io} + P_{io}W_{cpu} \tag{7.9}$$

For the optimal balance (β_{io}^{opt}) between CPU speed and I/O bandwidth, Equation 7.9 should equal to 0. Then, solving for β_{io} and replacing it with the workload and

technology-cost balance terms mentioned in Table 7.1 we get:

$$\beta_{io}^{opt} = \sqrt{\frac{\gamma_{io}}{\delta_{io}}} \quad (7.10)$$

Similarly, the optimum balance (β_{mem}^{opt}) between CPU speed and size of DRAM can be derived as:

$$\beta_{mem}^{opt} = \sqrt{\frac{\gamma_{mem}}{\delta_{mem}}} \quad (7.11)$$

Equation 7.10 and 7.11 show the contribution of application balance and cost balance towards optimal system balance.

7.4.4 Observations and Inferences

Gray’s law is a special case of the model when the cost balance equals the inverse of the application balance (i.e., the application’s CPU I/O and, memory requirement exactly balance the contemporary cost of the hardware).

Figure 7.1 compares the model with Amdahl’s second law and Gray’s law. The horizontal x-axis shows the different types of application balance. Value of $\gamma_{io} = 1$ presents an I/O- as well as a CPU-intensive application where Gray’s I/O law and Amdahl’s I/O law both intersect. Likewise, $\gamma_{mem} = 1$ presents a memory and CPU-intensive applications. It can be observed that the model suggests using less DRAM than suggested by Gray’s memory law. However, the model yields higher values for system’s I/O bandwidth comparing to Gray’s I/O law. This is because the current price of magnetic disk or SSD is much lower than that of DRAM.

It can be noticed that lower balance ratio between per-GBPS-I/O-cost to per-GHz-CPU-cost leads to higher balance ratio between system-I/O-GBPS to system-CPU-GHz. One straightforward interpretation for this observation is that if per-GBPS-I/O-cost starts decreasing faster than the per-GHz-CPU-cost, designers should increase the system-I/O-GBPS of a single server to achieve the new I/O

balance ratio (GBPS/GHz). However, instead of scaling up a single server in terms of storage, designers can scale out in terms of processor. That is, reduce the number of processor in a single server and add more servers with the same new system balance ratio. That is, both scaled out and scaled up architecture can produce optimal price-performance-ratio if the proper balance is maintained.

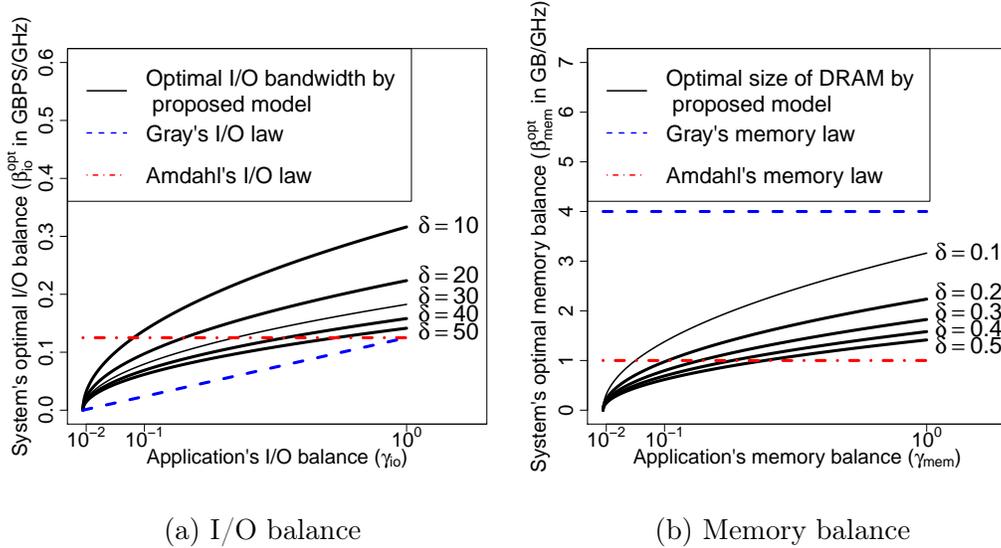


Figure 7.1: Change in system's optimum I/O balance (β_{io}^{opt}) and memory balance (β_{mem}^{opt}) as a function of application balance (γ_{io} and γ_{mem}) for different cost balance (δ_{io} and δ_{mem}).

7.4.5 Notes on Different Types of I/O

Since the model is generalized for different types of I/O, system designers should be careful about the following issues to maintain the system balance (i.e., to achieve the recommended value of β_{io}):

1) Sequential vs Random I/O: System designers should be careful about the application characteristics. An application with frequent random I/O (e.g., shuffle phase of a Hadoop job) will get benefit from SSD.

2) Disk Controller: Aggregate I/O bandwidth of all the disks attached to a compute node should be less than or equal to the bandwidth of the disk controllers. Otherwise, disk controller will be saturated and the application cannot utilize the

full I/O bandwidth of all the disks. [71] demonstrated this issue.

3) Network I/O: The network I/O can be modeled similarly as the disk I/O model. However, system designers should be careful about network topology (e.g., FatTree, Clos, etc.) and the blocking ratio which can be changed manually affecting the cost per network bandwidth. Hence, to apply the proposed model for network I/O the impact of these factors should be eliminated. That is, given the same topology and blocking ratio system designers can easily calculate the optimum ratio between CPU speed and network bandwidth using the proposed model.

7.4.6 An Example of Building a Balanced Cluster

This section demonstrates a real example of how to apply the model to build a cost-effective, balanced cluster. To reflect today's data-, compute-, and memory-intensive scientific applications, it is considered that the work done by the CPU, I/O and memory subsystem (i.e., W_{cpu} , W_{io} , and W_{mem}) are equal for that application. That is, using the notation in Table 7.1, the application balance can be written as, $\gamma_{io} = \gamma_{mem} = 1$

The unit price in Table 7.2 shows the current price trend for different processor, storage and memory alternatives in their corresponding unit. Intel Xeon processor is considered. As it can be seen in Table 7.2, the cost per MBPS of sequential I/O for both HDD and SATA-SSD is almost similar irrespective of change in storage technology provided the same storage space per disk. Whereas, the I/O bandwidth cost started reducing significantly with NVMe SSD. The cost per GB of DRAM is increased almost double from DDR2 to DDR3.

The average cost of each hardware component is calculated from the available list (Table 7.2) in terms of their corresponding unit price. For example, two different Xeon processors, E5-series and D-series have been shown in Table 7.2. Their respective unit prices are \$42/GHz and \$54/GHz. Hence, in this example, average unit cost of the processor is selected as \$48/GHz. Similarly, the cost of the DRAM

Table 7.2: Cost of different hardware components

Hardware components	Cost(\$)	Unit Price
Intel Xeon 64bit 2.6 GHz E5 series (8-cores) processor	1760	\$42/GHz
Intel Xeon D-1541	650	\$54/GHz
Western Digital RE4 HDD (120MBPS), 500GB	132	\$2252.80/GBPS/TB
Western Digital VelociRaptor HDD (150MBPS), 600GB	167	\$1900.09/GBPS/TB
Samsung 840Pro Series SATAIII SSD (400MBPS), 512GB	450	\$2250.00/GBPS/TB
Samsung NVMe SSD PM953 (2GBPS), 950GB	450	\$242.52/GBPS/TB
Samsung DDR3 16GB memory module	159	\$10/GB
32GB 1600MHz RAM (decided by Dell)	140	\$4.37/GB

is calculated as \$7.20/GB. To eliminate the impact of storage amount while calculating the I/O bandwidth cost, the unit price of a disk has been calculated in terms of cost per GBPS per TB. That is, the cost per GBPS is calculated using the same storage capacity (1TB) for each disk. This way, the average I/O bandwidth cost is \$1661.35/GBPS.

Next, the I/O cost per GBPS and DRAM cost per GB is calculated by the CPU cost per GHz to get the cost balance for I/O and memory respectively. Using the notation in Table 7.1, cost balance for this example can be written as: $\delta_{io} = 34.61$ and $\delta_{mem} = 0.15$.

Finally, for $\gamma_{io} = \gamma_{mem} = 1$ (i.e., CPU-, I/O- and memory-intensive application), using Equation 7.10 and 7.11 we get $\beta_{io}^{opt} = 0.17$ and $\beta_{mem}^{opt} = 2.7$.

7.5 Experimental Testbeds: Critical Analysis of Architectural Balance

Table 7.3 shows the overview of our experimental testbeds. The first one, SuperMikeII represents a traditional HPC cluster. SwatIII represents a regular datacenter. The last one, CeresII, is a novel hyperscale system, based on Samsung MicroBrick. In this section we characterize all these cluster with respect to our

model.

Table 7.3: Experimental testbeds

Resources	SuperMikeII	SwatIII	CeresII
Processor	Two -core SandyBridge Xeon	Two 8-core SandyBridge Xeon	One 6-core Xeon
CPU core speed	2.6GHz	2.6GHz	2.00 GHz
#Cores/node	16	16	6
CPU-Speed/node	41.6GHz	41.6GHz	12gHz
Disk/node and type	1-HDD (SATA)	4-HDD (SATA)	1SSD (NVMe)
Seq I/O Band- width/disk	0.15GBPS	0.15GBPS	2.00GBPS
Seq I/O Band- width/node	0.15GBPS	0.60GBPS	2.00GBPS
DRAM/node	32GB	256GB	64GB
β_{io}	0.003	0.015	0.166
β_{mem}	0.77	6.15	5.33

7.5.1 SuperMikeII (Traditional HPC Cluster)

This LSU HPC cluster offers a total of 440 computing nodes. However, a maximum of 128 can be allocated at a time to a single user. SuperMikeII has two 8-core Intel Xeon E5 series processor per node thus offering huge processing power. However, each SuperMikeII node is equipped with only one HDD (Western Digital RE4), thus limited in terms of I/O bandwidth. Also, each SuperMikeII node has only 32GB DRAM (Dell). As a result, SuperMikeII has $\beta_{io} = 0.003$ and $\beta_{mem} = 0.77$, both a magnitude smaller than the optimum produced by the model for a data-, compute- and memory-intensive application as shown in Equation 7.10 and Equation 7.11. Using the plot shown in Figure 7.1 (or using Equation 7.10 and 7.11 with SuperMikeII hardware cost shown in Table 7.2) SuperMikeII provides optimal price-performance-ratio for those applications where $\gamma_{io} = 0.0005$ or $\gamma_{mem} = .06$. Hence, it can be said SuperMikeII can provide cost-optimized performance for traditional compute-intensive applications such as supercomputing simulations, astrophysics calculations where γ_{io} has the order of 10^{-3} [94].

7.5.2 SwatIII (Existing Datacenter)

This Samsung datacenter has 128 nodes. However, maximum 16 nodes are used for the experiments. Unlike SuperMikeII which has only one HDD per node, SwatIII uses 4HDDs (Western Digital VelociRaptor) per node using JBOD (Just a Bunch of Disk) configuration while using the same processors (i.e. two 8core Intel Xeon E5 series) as SuperMikeII. Since the I/O throughput increases linearly with the number of disks, SwatIII's $\beta_{io} = 0.015$ is higher than SuperMikeII but lower than the optimum produced by the model for an I/O- and compute-intensive application (Equation 7.10). On the other hand, each SwatIII node has 256GB DRAM (Samsung DDR3), thus achieving a very high value for $\beta_{mem} = 6.15$. It is worth noticing that β_{mem} of SwatIII is even higher than the optimum produced by the model (Equation 7.11). Using Figure 7.1 (or using Equation 7.10 and 7.11 with SuperMikeII hardware cost shown in Table 7.2), it can be shown SwatIII can produce cost-optimized performance when $\gamma_{io} = 0.01$ and $\gamma_{mem} = 1.47$. That is, SwatIII can be a good choice for moderately I/O-intensive applications and for memory-intensive applications such as in-memory NoSQL. However, SwatIII may show worse price-performance-ratio for many of the modern I/O-intensive big data applications.

7.5.3 CeresII (MicroBrick-based Hyperscale System)

CeresII is a novel hyperscale system, based on Samsung MicroBrick with a maximum of 40 nodes available to us. Each MicroBrick (or simply a compute server) of CeresII consists of a 6core Intel Xeon D-1541 processor with a core frequency of 2GHz, one NVMe-SSD (Samsung PM953) with an I/O bandwidth of 2GBPS, and 64GB DRAM (Samsung DDR3). β_{io} of CeresII is 0.17 which is same as the optimum calculated by the model in Equation 7.10. On the other hand, β_{mem} of each CeresII module is 5.33. Although it is higher than the optimal, it is less than SwatIII. Thus, CeresII is the most balanced cluster among all the

available resources and it is expected to get the best cost to performance for today's I/O-, compute- and memory-intensive applications.

7.5.4 The Relation between Cluster Balance and Cluster Capability

The I/O and memory balance terms (β_{io} and β_{mem}) indicates the level of contention for I/O devices and memory subsystem respectively. The ratio between β_{io} (or, β_{mem}) of two different clusters can indicate their relative level of contention in I/O subsystem (or memory subsystem).

As a concrete example, let us consider CeresII (6 cores, 1 NVMe SSD per node, and $\beta_{io} = 0.166$) and SuperMikeII (16 cores, 1 HDD per node, and $\beta_{io} = 0.003$). For their corresponding value of β_{io} it can be said CeresII is almost 55 ($0.166/0.003$) times more balanced, or more powerful than SuperMikeII. It can be interpreted as CeresII has 55 times less I/O contention compared to SuperMikeII. Consequently, to achieve an optimized performance as CeresII, SuperMikeII needs almost 55 HDDs per node if those same 16 cores are used. This corollary of the proposed model can also be verified with other well-known cluster architectures such as, GrayWulf [105] which won the storage challenge in SC-08 with 8 cores and 30 HDDs per node (comparing to 16 cores and 55 HDDs per node as predicted by the model).

β_{io} and β_{mem} can also be used to determine the scalability across different cluster architectures more accurately. The speedup can be determined using Universal Scalability Law [106] by Neil Gunther, $S(p) = \frac{p}{1+c_1((p-1)+c_2(p-1))}$ Where p is the total number of processors. $S(p)$ is the speed up. c_1 is the level of contention and c_2 is the coherency delay. In absence of any coherency delay (i.e., $c_2 = 0$), the relative contention of two systems can be easily derived using the ratio of their corresponding β_{io} or, β_{mem} to get a better estimate of scalability. However, the detailed analysis of scalability of a data-intensive application, and modification of the corresponding laws should possibly be the focus of a different work.

7.6 Cluster Evaluation Methodology

7.6.1 Hadoop Configuration Overview

To evaluate different clusters Cloudera-Hadoop-2.3.0 and Giraph-1.1.0 are used. The Hadoop and Yarn parameters are set such that the application can make use of maximum available processing speed, I/O bandwidth, and DRAM. For example, all the compute cores are used for YARN where each Hadoop (and Giraph) worker uses one core. On the other hand, keeping 10% of DRAM for system use, the rest have been divided among all Hadoop workers equally. For Giraph, enough DRAM has been used to accommodate the entire graph structure in memory and always avoided out-of-core execution. The HDFS data (with replication factor 1) was spread and the shuffled data among all the disk(s) in the node using default scheduling of Hadoop.

7.6.2 Benchmark Application Characteristics

Table 7.4 summarizes the details of the benchmark applications: TeraSort, WordCount, and a real world genome assembly application described as follows:

7.6.3 TeraSort

The map phase of TeraSort samples and partitions the input data based upon only first few characters. The reduce phase then uses the quicksort algorithm to sort each of the partitions locally. The map phase of TeraSort is CPU-intensive as it reads only the first few characters of each row in the input dataset to generate the key (called sample-key) for each data. However, based on the data size, the reduce phase can be severely I/O-intensive.

7.6.4 WordCount

The map phase of WordCount parses the input dataset line by line to extract each word. Each word is emitted with a 1 as its initial count, which is then summed up in the reduce phase to output its frequency. Since both the map and reduce

Table 7.4: Data size for different benchmark applications

Job name	Job Type	Input	Output	# jobs	Shuffle data	HDFS Data	Application Characteristics
Terasort	Hadoop	1TB	1TB	1	1TB	1TB	Map: CPU-intensive, Reduce: I/O-intensive
Wordcount	Hadoop	1TB	1TB	1	1TB	1TB	Map and Reduce: CPU-Intensive
Graph Construction (human genome)	Hadoop	452 GB (2B reads)	3 TB	2	9.9 TB	3.2 TB	Map and Reduce: CPU- and I/O-intensive
Graph Simplification (human genome)	Series of Giraph jobs	3.2 TB (1.5B vertices)	3.8 GB (3M vertices)	15	-	4.1 TB	Memory-Intensive

phase read the entire dataset sequentially just once, both phases of WordCount is CPU-intensive.

7.6.5 Genome Assembly

Accounting the scarcity of good big data HPC benchmark and relevant big dataset we use the genome assembler developed in Chapter 4 built atop Hadoop and Giraph as a real use case which represents many HPC/Datacenter applications.

1) The first stage of the assembler is a shuffle intensive Hadoop job representing an I/O-bound application. It scans through all genomic short reads (lines containing A , T , G or C only) and divides them to a smaller fragment of length k , called k -mer. The map phase emits two consecutive k -mers as intermediate key-value pairs representing a vertex and an edge from it. Reduce phase aggregate all edges from each vertex to write the entire graph structure on HDFS.

2) The second stage of the genome assembler represents terabyte-scale graph processing which is the core part of many HPC problem. In this phase, each of the linear chains in the graph structure is compressed to one vertex. Then all the tip and bubble structures of the graph are removed as those are introduced because of sequencing error.

7.6.6 Data Size

For both TeraSort and WordCount, a 1TB random dataset is generated as the input. The shuffle and output data size of TeraSort is also same as its input (i.e., 1TB in this work). The output of WordCount may vary based upon the frequency of different words in the randomly generated dataset. However, it is closed to 1TB.

For the genome assembly benchmark application, this thesis uses a large human genome dataset (452GB), openly available in NCBI website ¹ with accession number SRX016231. The corresponding graph size is 3.2TB. The Hadoop stage of the assembly application is severely shuffle-intensive. The temporary shuffle data is almost 21-times more than the input size.

7.7 Results and Discussion

7.7.1 Evaluation Results of TeraSort and WordCount

Figure 7.2 compares the relative merit of all the three cluster architectures (i.e. SuperMikeII, SwatIII, and CeresII). To show the balance between performance and economy, that many resources are used in each cluster which keep the total cost same across all the clusters. Table 7.5 shows the available resources for all three clusters while keeping the total cost same across all the clusters. The cost of 16 nodes of SuperMikeII has been used as the baseline. Then, this baseline-cost is divided by the cost of each node in SwatIII and CeresII to count the number of nodes to be used in these two different clusters.

¹<http://www.ncbi.nlm.nih.gov>

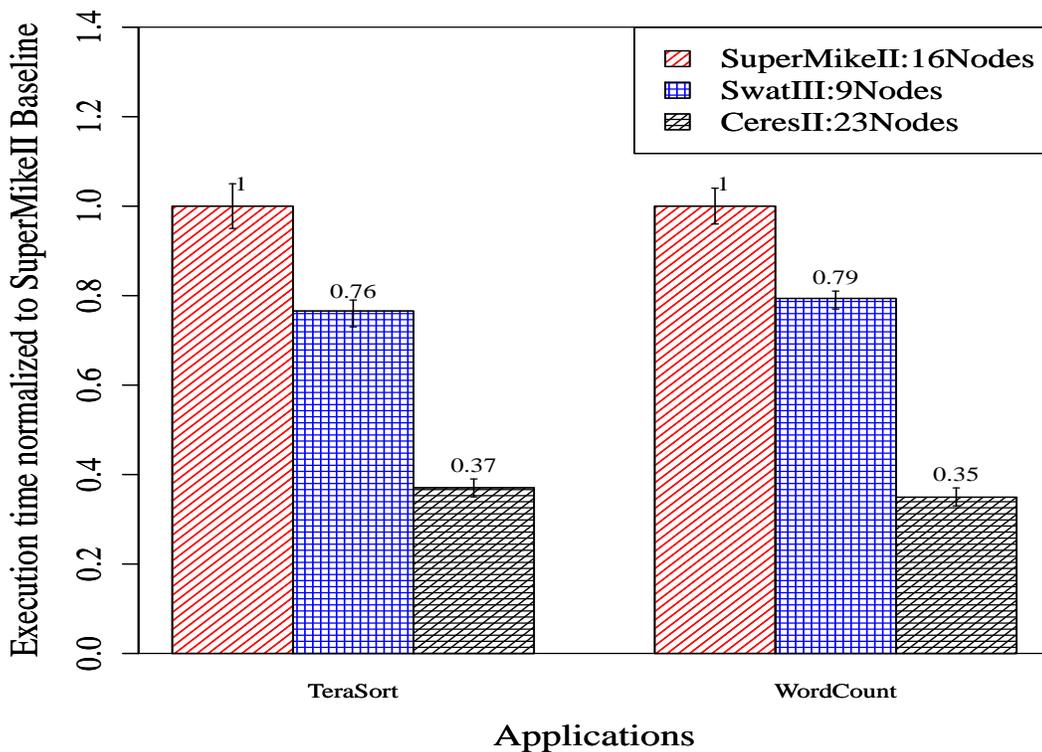


Figure 7.2: Execution time (normalized to the baseline) of TeraSort and WordCount over different cluster architectures keeping the total cost of each cluster same.

Table 7.5: Resources in each cluster architecture used for TeraSort and WordCount

Cluster Configurations	SuperMikeII	SwatIII	CeresII
Total cost (\$)	60864	60864	60864
Cost/nodes (\$)	3804	6911	2700
#Nodes	16	9	23
Total processing speed (GHz)	665.60	374.4	276.00
Total I/O bandwidth (GBPS)	2.4	5.40	46.00
Total storage space (TB)	8.00	21.60	21.85
Total DRAM Size (TB)	0.50	2.34	1.47

TeraSort and WordCount were executed in all three cluster configurations and measured their execution time. All results are the means of at least three runs of each application on each configuration. Figure 7.2 shows the results normalized to the SuperMikeII baseline. That is, the execution time of SuperMikeII is always

assumed as 1 and the other execution times are adjusted by multiplying them with similar fraction. CeresII, being closer to the optimum produced by the model performs significantly better than the other cluster architectures. The observations are as follows:

- 1) Comparing to the SuperMikeII baseline, for both TeraSort and WordCount, CeresII shows almost 65% improvement.

- 2) Comparing to SwatIII, CeresII shows almost 50% improvement in execution time for TeraSort and WordCount.

7.7.2 System Characteristics

To monitor the system characteristics the Cloudera-Manager-5.0.0 is used. The system characteristics are shown at the end of the chapter (Figure 7.3). The observations are as follows:

- 1) As shown in Figure 7.3a and 7.3b, for the compute-intensive WordCount application, on an average CeresII shows 20% better CPU utilization compared to SuperMikeII as the goal is to perform the task as soon as possible resulting in better price-performance-ratio. Whereas for TeraSort with an I/O intensive reduce phase results in almost 50% better CPU utilization in CeresII than SuperMikeII. Comparing to SwatIII, CeresII shows almost 15 to 40% better CPU utilization for TeraSort and WordCount respectively. Since the goal is to perform the task as soon as possible (to better utilize the system's cost), CeresII is considered as the best architecture.

- 2) Figure 7.3c and 7.3d shows the reason behind the better CPU utilization of CeresII. Since the architectural balance of CeresII closely resembles the optimal β_{io} produced by the model, CeresII shows almost negligible I/O wait (less than 2%) for any of the applications. On the other extreme, SuperMikeII with only one HDD results in extremely low β_{io} , consequently shows highest I/O wait among all the cluster architecture. SwatIII with four HDD per node shows an I/O wait lower

than SuperMikeII but significantly higher than CeresII.

3) Figure 7.3e and 7.3f compare the I/O throughput of all the clusters. CeresII with the most optimal β_{io} shows the highest I/O throughput and SuperMikeII with the lowest β_{io} shows the lowest I/O throughput among all the clusters. SwatIII lies in between these two extremes as its β_{io} lies between them.

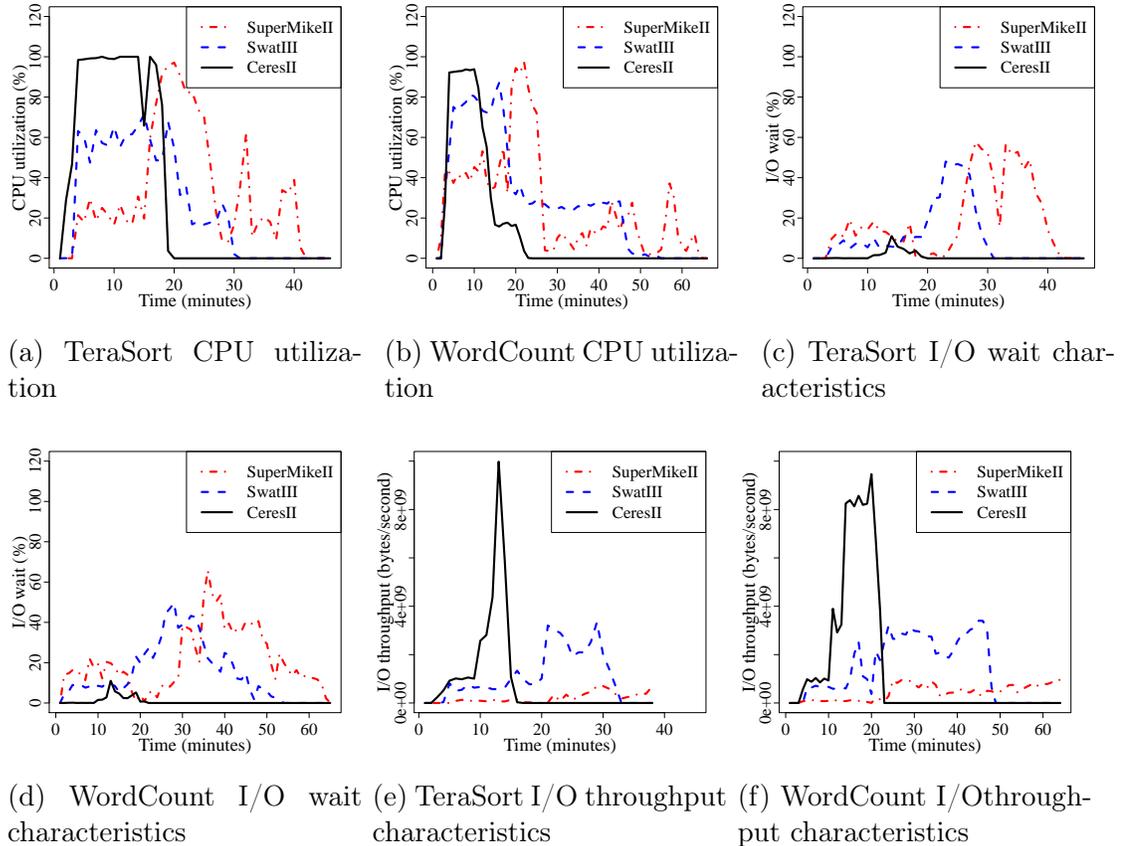


Figure 7.3: CPU and I/O characteristics of each node of different cluster architectures for TeraSort and WordCount benchmark

7.7.3 Evaluation Results of Large-Size Human Genome Assembly

To assemble the large human genome (452GB), the maximum available resources in each of the clusters are used to accommodate the huge amount of shuffled data (9.9TB) and the graph data (3.2TB). That is, all 128 nodes of SuperMikeII, 16 nodes of SwatIII, and 40 nodes of CeresII were used for this application. Table 7.6 shows the cost and the configurations of the clusters. Figure 7.4a and 7.4b

Table 7.6: Maximum available resources in each cluster architecture (used for large human genome assembly)

Cluster Configurations	SuperMikeII	SwatIII	CeresII
Total cost (\$)	486912	110576	108000
Cost/node (\$)	3804	6911	2700
#Nodes	128	16	40
Total processing speed (GHz)	5324.8	665.6	480.00
Total I/O bandwidth (GBPS)	19.2	9.60	80.00
Total storage space (TB)	64.00	38.40	40.00
Total DRAM Size (TB)	4	4	2.56

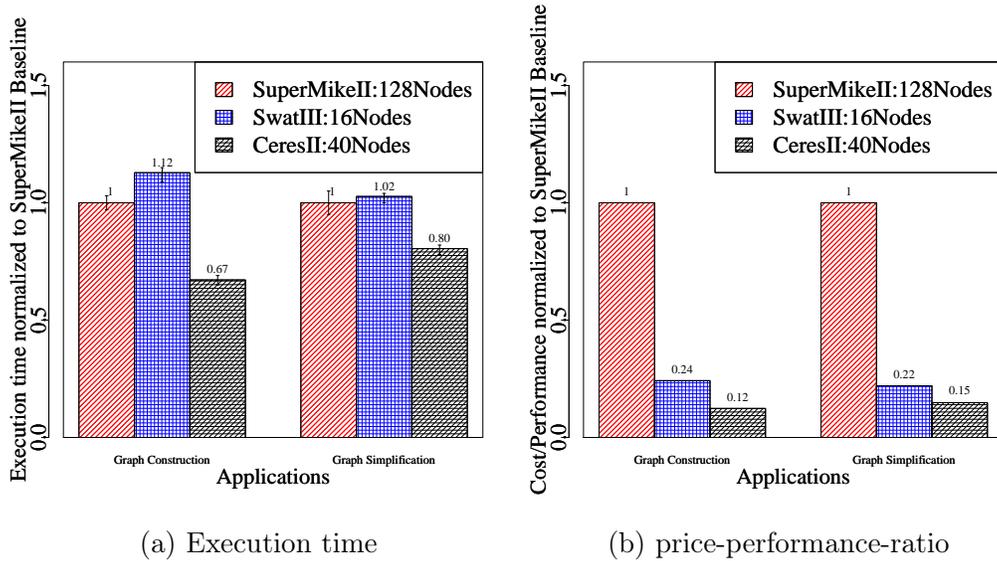


Figure 7.4: Performance of different cluster for large size human genome assembly (normalized to 128 nodes of SuperMikeII).

show the corresponding execution time and the price-performance-ratio (i.e., cost \times execution-time) respectively for the Hadoop and Giraph stages of the human genome assembly. The results are as follows:

1) CeresII, even with almost 90% less processing speed than SuperMikeII across the cluster, outperformed it by almost 88% in terms of price-performance-ratio. In Giraph stage the corresponding gain in 85%. In terms of execution time CeresII gains almost 30% and 20% respectively over SuperMikeII.

2) Comparing to SwatIII, the processing power of CeresII is 72%. However, due to the optimal architectural balance, CeresII shows almost 50% and 30% improve-

ments over SwatIII in terms of price-performance-ratio for Hadoop and Giraph respectively. For execution time, those gains are 50% and 20%.

7.8 Conclusion

Big data needs big resources. With increasing popularity of data-driven research, it is obvious that providing more resources (CPU, I/O bandwidth, DRAM, etc.) will provide more performance. So, the major challenge is now in providing expected performance in reduced cost. This thesis makes an initial attempt to analytically resolve the performance and cost conundrum prevalent in big data cyberinfrastructure.

The model also provides a new metric to show the capacity of the HPC clusters. For I/O and memory-bound applications, β_{io} and β_{mem} can provide a better and easy-to-use alternative to FLOPS which shows only the CPU capacity.

In this initial attempt the thesis focuses on simplicity of the model to make it useful for practical purposes (e.g., investing for a new cluster with limited information on application characteristics). However, more subtle parameters (e.g., CPU multi-threading, I/O latency, etc.) can be added to improve its accuracy.

Chapter 8

High Throughput Transaction of Big Biomedical Data with Blockchain and P2P Storage

This chapter introduces a decentralized medical data interoperability system, SwarMed which leverages Blockchain technology, developed with the Ethereum ecosystem. To address the solution for managing large-scale data, common and crucial for medical records but lacking with the vanilla Blockchain technology, the system is built upon Swarm, a p2p storage system and then strengthened and secured with Ethereum Blockchain. Along with a proper consent management for sharing sensitive data, this chapter also developed an indexing mechanism over the immutable storage of Swarm to achieve high-throughput while sharing millions of patient records and images among multiple parties.

SwarMed achieved a high throughput of 250K medical records per second over a private network constructed over LSU-HPC cluster. This high throughput is 9x more comparing to conventional way of using p2p storages in conjunction with Blockchain. This high throughput enables the patients to get real-time access to his comprehensive medical history and scientists to gain real-time access to different medical data for collaborative research complying to the constraints posed by existing laws.

The system level analysis over different design alternatives over different transfer- and storage-architectures shows that, p2p storage platforms automatically provide significantly better scalability over traditional HTTP with increasing number of clients. Swarm provides 2x more I/O throughput and 10x less latency than IPFS, another p2p storage making it a better choice for decentralized big data transfer.

8.1 Introduction

Blockchain has started taking the world by storm. Soon after its development as the underlying architecture for Bitcoin, the trust-less, decentralized concept of the Blockchain was recognized as having broader value beyond an alternative form of currency. Starting from health care to IoT, from education to translational research, the fundamental model of trust for information sharing is rapidly changing. Deviating from the traditional centralized architecture e.g., client-server model, cloud, etc., many applications have started using current state of the art, decentralized Blockchain platforms, such as Ethereum, Bitcoin, Hyperledger etc., not only for financial transaction but also for large-scale data transaction.

Blockchain received a significant attention in the healthcare sector. Millions of patients' records if analyzed properly, can have tremendous business and/or research implications. However, the records are fractured over thousands of care providers' site and protected by strict regulations such as HIPAA, COPAA, CURE, etc. As shown in this chapter, Blockchain with its smart contracts and decentralized trust model can provide effective and secure solution to this problem.

However, the technology (Blockchain) even with its tremendous security promises has severe performance bottleneck especially in terms of size of data that it can handle per transaction hindering its large scale adaption in the healthcare domain. The size of each block is typically limited to a few MB only irrespective of the underlying Blockchain network (e.g., Ethereum, Bitcoin, Hyperledger, Parity, etc.). The limitation in the data handling capacity also impacts the overall performance of the network in terms of throughput and latency. Complicating the scenario, the size of the block shares a complex relationship with the existing economy which makes it hard to change the block size especially in the public Blockchain networks.

As a result, building an efficient and cost-effective infrastructure for transaction of big data in a decentralized and trustless manner became more challenging, albeit

crucial in the domain of health care. Consequently, this started opening new opportunities for the researchers as well as industry leaders as trillions of dollars are already invested in on the promising decentralized technologies. Furthermore, the technology is evolving at an unprecedented rate giving birth to an entirely new ecosystem centering different Blockchains such as, Ethereum, Hyperledger, etc. For example, Swarm [19] and Inter Planetary File System (IPFS) [107] are peer-to-peer storage systems that can be used for off-chain data transaction addressing the limitation of the block size. Whisper [108] has been emerged as a communication protocols for decentralized applications to communicate among peers.

Hence, there is a growing interest in all the four communities, including scientists, health care providers, patients, and commercial Blockchain service providers (e.g., Ethereum community, Bitcoin, IBM, etc), to develop cost-effective, high-throughput infrastructure for medical data interoperability that will drive the next generation healthcare research involving huge amount of big data while at the same time utilizing security promises of Blockchain. Millions of dollars are being spent in programs, such as ONC's nationwide challenge [109], where several academic organizations and industry leaders collaborated and competed to address the challenges involved in developing a novel distributed and decentralized cyberinfrastructures for healthcare data interoperability.

Despite this growing interest in all the communities, there is a limited understanding of how the different types of data storage and transaction mechanism in conjunction with Blockchain impact the performance and scalability of the system when applied to real-world application involving large data. Furthermore, because of a scarcity of open source software tools, limited amount of research have been performed over the system characteristics of the Blockchain-based system which can help developers to identify the bottlenecks and improve the architecture accordingly.

This work first proposes secured, trustless, decentralized framework for medical data interoperability with proper consent among patients, providers and third-party stakeholder. However, for this paper, this chapter focuses mainly on the performance issues at the system level only. The framework is based on Ethereum Blockchain and makes use of its Web3 APIs for the decentralized application (dApp). The dApp uses a mix of an on-chain and an off-chain mechanism to transfer the data.

To transfer the large number of EMRs, EHRs, images (such as x-ray images, mammogram images, etc). this chapter developed an efficient indexing mechanism over Swarm Swarm, a peer-to-peer data storage. The personal index file of a patient grows in size as more medical records are added. The Ethereum transactions (tx) stores only the pointer to the index file limiting the growth of block size, thereby improving the throughput of the entire system. The architecture achieved a constant throughput of more than 250K records per second even though more data is added. Given the size of an address pointer to the off-chain data is 32bytes, in a realistic scenario, this throughput is more than 9x better comparing to the conventional way of storing all the address pointers for the data on the chain.

The rest of the chapter is organize as follows: Section 8.2 describes the prior efforts to the current work. Section 8.3 provides the background of different technologies. In Section 8.4, the thesis discusses the high throughput architecture of SwarMed. Section 8.5 evaluates the proposed architecture with several different alternatives. Finally, Section 8.8 concludes the chapter.

8.2 Related Work

Earlier studies, as well as the prior experience, show that Blockchain and its decentralized architecture can be helpful in improving the IT infrastructure of different domains including health care [110], HPC [?], IoT [111], education [112] etc. Among all, the healthcare sector gained the maximum focuses in the

last 2 years especially in response to ONC's challenge (ONC)[109]. Following the challenge, a plethora of works has been done finding opportunities for applying Blockchain technology to health care to make health information exchanges (HIE) more secure, efficient, and interoperable.

Blockchain offers an umbrella of technologies related to data security and privacy including decentralized management, distributed ledger, immutable audit trail, data provenance etc. presenting opportunities for disruptive innovation. [113, 114], etc. envision how each of these technologies can improve the existing infrastructure of EMR-management and insurance claim processing, the two major foundations of the US healthcare system. [115] envisions a more transparent treatment process by utilizing the immutable audit trail of Blockchain especially fighting against counterfeit medicine. [116] envisions an improved relationship between patient and physician by shifting the trusted intermediary role away from the hospital and into the Blockchain. Although these visions are laudable and clearly show the Blockchain's opportunities in the medical informatics, the underlying technology (i.e., Blockchain) has been changed at an unprecedented rate in the last couple of years. A considerable number of Blockchain (e.g., Ethereum, Parity, etc.) has been introduced focusing on different aspects and trade-of related security, privacy, and performance. [117] wisely captures these changes by qualitatively evaluating the privacy and security concerns of health care from the perspective of all different types of Blockchain including public, permissioned and private Blockchain.

Moving a step forward, [118, 119, 120], etc. discuss the basic building blocks and a detailed design of a Blockchain-based infrastructure to develop, govern, and operate a network with the security necessary for many demanding use cases in the regulated domain of health care. [121] narrowed down the problem and discusses the opportunities of Blockchain in predictive medicine. [110] on the other hand

addresses the issues in the fractured medical record over different providers' site. [110] proposes a prototype to enable the patient to access his own medical records fractured over multiple providers. The prototype has a serious privacy issue. The authors proposed to use patient's medical record to provide an incentive to the miners as it was the only feasible option at the time of writing that paper. [122] resolves the privacy issues by assigning the sensitivity score and controlling access to the records according to that score. However, the framework uses a centralized database to hold the sensitivity information which leads to a central point of failure resulting in a denial of service. the removal of the centralized database will result in data replication over multiple databases (located at different providers' site). Based on the number of patient and the granularity level of the records the design can lead to a significant trade-off between space requirement and security. [123] proposes a framework to share the mobile healthcare information from the patient's wearable devices to the healthcare providers. However, the architecture facilitates the data flow between patients and the corresponding providers only. Consequently, it does not contribute much towards the interoperability issues in the healthcare domain.

Although common and crucial for medical records, data management capability is severely lacking in the vanilla Blockchain technology. Consequently, the challenges involved in big data transfer and interoperability is not effectively addressed in the framework discussed above. However, peer-to-peer storage technologies such as Swarm, IPFS, etc. are gaining popularity in recent years for decentralized data transmission. For example, [111] uses IPFS to transfer the IoT data.

These peer-to-peer file system with decentralized security promises of Blockchain can resolve many of the core issues in medical record management which is growing at an exponential rate, however, in a fractured manner over thousands of providers' site [110]. Use of these off-chain file system will not only make the sharing process easier but also improves the performance issues in the existing framework while

preserving the decentralization philosophy. [124], in their benchmark study also realized the need for off-chain data storage for data transfer over Blockchain. However, their study did not focus on big data transfer. [110] also separated the storage from the Blockchain using traditional database system in each of the provider's site while storing all the pointers in the Blockchain. However, the storage capacity of each individual transaction (and smart contract) is severely limited. The performance of entire system goes down even for a small increase in data byte stored as shown in this work. To remedy this, the thesis came up with an indexing mechanism on decentralized p2p storage, so that the system can take the advantage of off-chain storage systems as well as get the full performance from Blockchain.

8.3 Background

8.3.1 Blockchain: Ethereum and Smart Contract

The Blockchain technology became popular predominantly with its application in Bitcoin [16]. A chain of blocks representing the history of transaction events is established and maintained in a fully decentralized fashion abandoning the norm of a financial system relying upon a small number of centralized organizations such as banks and governments. The proposed community model operates with the mechanisms such as a consensus scheme of proof of work, secured methods with cryptographic techniques, a privacy policy in public space, incentive-based mining, and the distributed ledger system, i.e. Blockchain. The main nature of Blockchain runs over peer-to-peer networking, making sure that each node keeps its own copy of the chain and updated it together to add a new block. Soon after recognizing the potential of this concept of decentralized autonomous organization (DAO) applied for the Bitcoin cyber currency system, a significant amount of interest have been exploded for other applications. Among them, Ethereum [17], BigChainDB [125], HyperLedger [18], etc. have emerged providing Blockchain-based platform with

which a variety of different applications can be developed, as demonstrated by the current work.

The Ethereum community runs a fully open public network, called the main network, for which the cyber currency is Ether. Unlike Bitcoin, Ethereum is, in fact, a platform providing programming tools and utilities (<https://github.com/ethereum/>). Using this resource, for example, even a private network separated from the public network can be constructed. Along with the main network, for the developmental purpose, another public network called Testnet is also operated by the Ethereum project. The core of this platform is an ecosystem for developing DApps (Decentralized Applications) whose main purpose is to manage and execute Smart Contract or contract hereafter. Contracts are the major entities to build business or project logic on the top of the Blockchain network. Building a DApp requires two components, the front-end, which contains user interface components as well as APIs for interacting with contracts, and the back-end is the component capable of executing contracts. The front-end of Ethereum DApp is basically a web application written with HTML/Javascript. The back-end contains EVM (Ethereum Virtual Machine), which run bytecodes of contracts and is responsible for communicating with other nodes in the network using peer-to-peer networking.

8.3.2 Peer-to-Peer Storage: Swarm and IPFS

The P2P storage systems extend the concept of the Markle tree for storage providing an immutable, easy-to-audit, tamper-proof infrastructure for storing files. Like Blockchain, the solution is secured by Markle Hash. However, it is off-chain and designed for higher throughput compared to the smart contract-based storage of the main Blockchain. Furthermore, the Markle tree is constructed based on the actual content of the files (i.e. the data itself) providing a content-based hash to retrieve the data rather than an actual server-address in the traditional HTTP-based system. Consequently, the data can be downloaded from the nearest peer

of the P2P system, rather than a specific server located significantly many more hops apart in a traditional system. It results in a significantly higher throughput and scalability during the data download. In addition to that, these P2P storage systems are automatically scalable as each of the peer or the clients works as a storage server also and they are committed by means of some incentive mechanism of the system.

Both Swarm [19] and IPFS [107] leverage Merkle tree to distribute the data over peers in a content addressable fashion. However, in the lower level of implementation, the two systems use different types of storage technology, network communications, and peer management protocol. For example, Swarm storage system is basically an immutable content addressed chunk-store whereas, IPFS uses a distributed hash table (DHT) for storage purpose. For peer management, Swarm uses Ethereum-based protocol (commonly known as devp2p) whereas IPFS uses a BitTorrent-based protocol (commonly known as libp2p). Both Swarm and IPFS implement a key-based routing based on xor logarithmic distance. However, IPFS uses iterative lookups of peers at the originator of a request (such as, the node where a write/upload operation is performed) relying on a larger pool of peers. Swarm on the other hand, recursively outsource the steps for looking up the peers using only a smaller pool of active connections.

8.4 SwarMed Architecture

8.4.1 Interoperability Model

Figure 8.1 shows the interoperability model of SwarMed. In this model, the consensus layer and the storage layer are decoupled adding a 3-phase privacy and security to the system. In the system, the consensus layer is basically a consent management system which spans globally across all the participants taking care of legal issues. The actual data layer, on the other hand, forms small groups adding

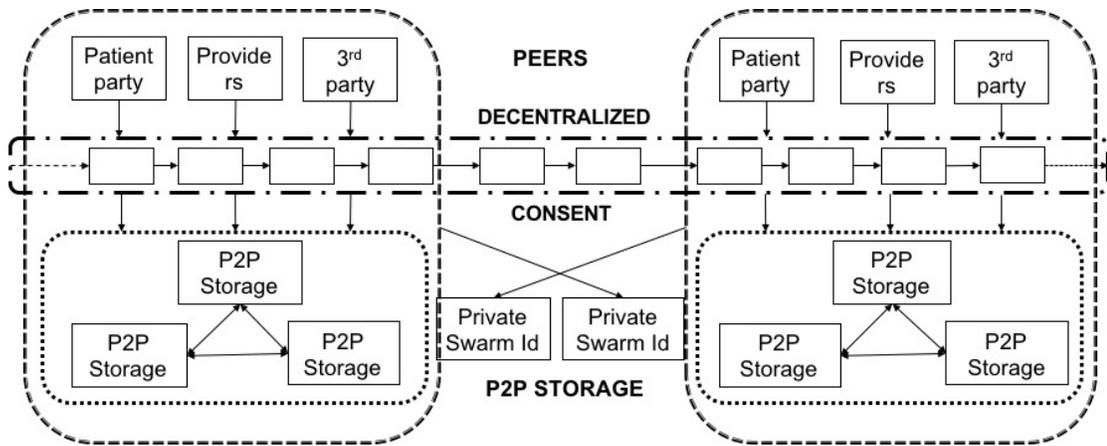


Figure 8.1: Interoperability model

an extra layer of authentication. That is, to access the records a user needs to have all the three, a valid consent, the private Swarm id and the actual Swarm hash of the data. For example, tightly coupled organizations such as a medical school and the scientists in the corresponding university and their frequent visitors (patients) may form a group and share the data among themselves when a proper consent is given. However, members from other groups should pass through the authentication layer of this group in order to access the actual data.

8.4.2 Software Architecture

Figure-8.2 shows the architectural overview of SwarMed. In this proof-of-concept version, this chapter developed SwarMed as an interoperability layer isolated from the individual's database or file system. That means the user has the full control over how much and what types of data are to be shared.

SwarMed architecture has three different layers such as, Blockchain layer which takes care of the consensus protocol, peer-to-peer storage layer, and data management layer. Following is the description of each of the layers. For the sake of brevity, this work mainly focuses on the performance issues at the system level and avoid the intriguing details of the smart contracts that take care of the legal consents.

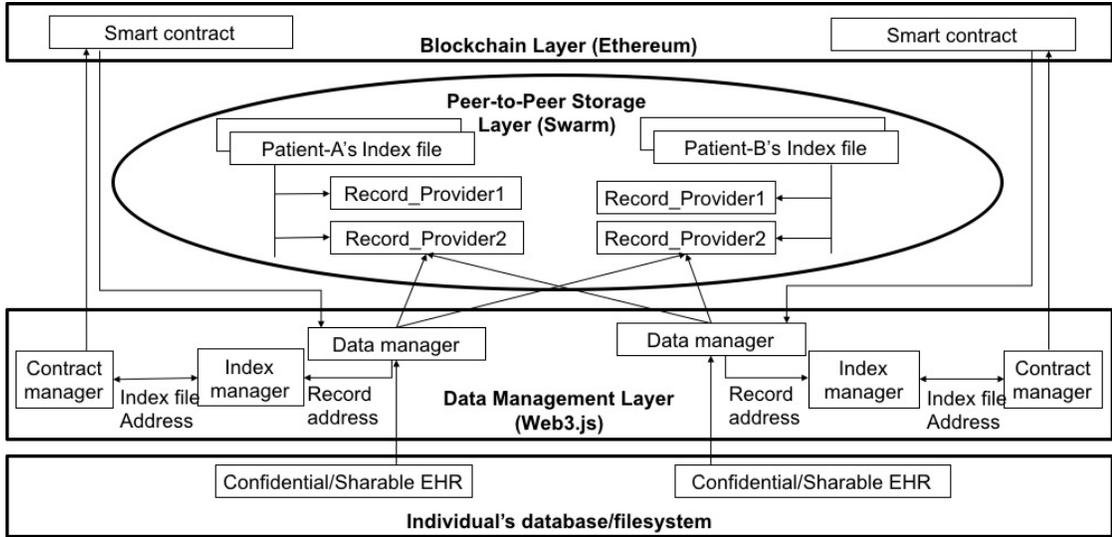


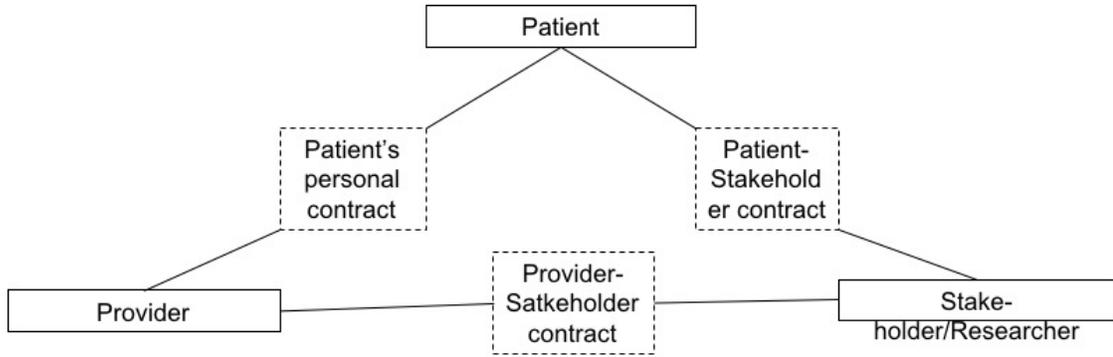
Figure 8.2: SwarMed's decentralized architecture

8.4.3 Blockchain Layer

This layer implements a set of smart contracts providing the full functionality required to join and participate in the Blockchain network. For the prototype implementation, this thesis uses Ethereum and the Web3.js client. While Swarm (the peer-to-peer data storage layer) acts as the main data storage layer in the prototype, Ethereum's smart contract provides the authentication and other logistic services to access those data. In fact, the smart contracts implemented here handles a broad set of tasks, such as connecting to the peer-to-peer network, encoding and sending transactions and keeping a verified local copy of the Blockchain.

This chapter utilizes Ethereum's smart contracts to create consent among different parties to share the medical records Swarm network. For better understandability, the contracts are presented as the relation between the three entities of SwarMed: patient, provider and other third party stakeholders (e.g., scientists and researchers). Consequently, there are three different types of contract in the system as shown in Figure 8.3a

Figure 8.3b shows the minimal structure of a smart contract. In the broad level, all the logistics i.e., data ownership, an agreement for data sharing, etc. are



(a) Different types of smart contracts in SwarMed

SwarmAddressOfIndexFile: 0x12ab....
OptionalPermissionFlag: 0/1
<pre>FunctionToChange(_new SwarmAddressOfIndexFile, _new OptionalPermissionFlag) { if (legitimate user) SwarmAddressOfIndexFile = _new SwarmAddressOfIndexFile; OptionalPermissionFlag = _new OptionalPermissionFlag; }</pre>

....

(b) Minimal structure of a smart contract

Figure 8.3: Smart contracts

taken care of by these smart contracts by modifying the hash of the index file based on privacy and authorization rules. It is possible to add more rules to the contract complying with HIPAA and/or other privacy concern. The Blockchain transactions carry cryptographically signed instructions to manage these properties. The contract’s state-transition functions carry out policies, enforce data alternation only by a legitimate entity. There are three different types of contract as follows:

1. Patient’s Personal Contract: This contract represents the relationship between a patient and the providers. Once the patient is registered to the system, SwarMed issues a contract to the patient that can be shared with all the providers visited in the patient’s lifetime.

This contract contains a pointer to the index of the records that are uploaded to the Swarm cluster. For a patient, initially, the contract points to an empty

index file kept in Swarm. Gradually, the providers start uploading the patient's medical record to Swarm and subsequently add the indexes of the patient's medical record to the index file. As discussed earlier, since Swarm provides an immutable storage, in this context update means creating a new index file in a new address. This address is updated in the patient's personal contract.

2. **Provider-Stakeholder-Contract** This contract represents the relation between the care providers and other stakeholders or researchers. Like the patient's personal contract, this contract also contains a pointer to an index file kept in Swarm. However, this index file points to all the patients' record that the patient agrees to share with other third parties.

Essentially this contract has a permission flag indicating the patient's agreement on sharing the medical record. At any point of time patient can request to revoke permission. Consequently, the provider can delete the index of that patient's data and update the contract with a new index file following the similar process discussed earlier.

3. **Patient-Stakeholder-Contract** In SwarMed, a patient can share his own medical record directly with the stakeholders (or researchers). SwarMed did it using a contract between a patient and a stakeholder similar to that between the provider and stakeholder.

8.4.4 Peer-to-Peer Storage Layer

This layer stores the actual EHR data that are to be shared with the patient and other stakeholders. The patient or other legitimate stakeholders can access the data in Swarm using bzz protocol or its variant (e.g., bzzi and bzsr).

The major objective of using peer-to-peer storage is to provide a storage for big data that is DDOS-resistant and fault-tolerant. Data is stored in different nodes as small chunks and stay distributed. Consequently, there is no central target for the content attack which is crucial for sensitive medical information.

As discussed earlier, using Swarm this work separated the data layer from the logistics developed in Ethereum smart contract, thus enabling sharing of the huge amount of data which cannot be handled using Blockchain alone. In SwarMed, the data resides off-the-chain in its entirety and do not increase the block size at any time. This way SwarMed guarantees eliminating any negative impact (i.e., lower throughput or higher latency) on the Blockchain because of the larger size of the block. To do that SwarMed developed an indexing service over Swarm. SwarMed keeps only one Swarm address in the Ethereum contract which points to an index file kept in Swarm. This index file, in turn, contains the pointers to the raw medical records. This process is discussed in detail later in this section.

Furthermore, Swarm provides an immutable data storage, i.e., the data files uploaded once cannot be edited. By using this property, SwarMed enables an easy audit trail off the chain in case of a malicious attack or erroneous activities in the Swarm network so that the system can easily rollback to its previous correct state.

8.4.5 Data Management Layer

This layer consists of three different modules as follows:

1. **Data Manager:** This module provides the only access interface to the node's local database or file system. This module accommodates complex database queries to select the subset of data that the node wants to share with the other entities of the system. There are four distinct objectives of the data manager. Those are as follows:

First, it allows SwarMed to operate as a separate layer isolated from the actual medical database of the providers (or, any other parties). Thus data manager gives extra yet easy control to the individuals (i.e., provider, patient or researcher) on what data to share through SwarMed. Data can be chosen to be shared by the individual based on their confidentiality or privacy.

Second, the data manager can easily integrate any type of common data

model (such as PCORNet) to enable systematic analysis of disparate observational datasets which are chosen to be shared over SwarMed.

Third, data manager also allows the individuals to encrypt their data separate from Ethereum and Swarm. It is always possible to implement or integrate any kind of security mechanism inside data manager (such as public/private key cryptography) before sharing it with the peer-to-peer storage in the public domain. Thus the data manager can work as an extra layer of security between the individual database and the public domain storage space.

Finally, unlike existing prototypes which rely on pull model of data sharing, data manager enables push model for an improved user experience in Blockchain network. That means the required subset of data is prepared beforehand to reduce the overall latency. To do that, the data manager in a node always keep track of the contracts of that node and the data stored in that node. Whenever a new record is added to that node the data manager checks the existing set of contracts and upload the data to Swarm and handover the address (a Swarm-Hash) to the index manager for further processing.

2. Index Manager: Index manager implements a query interface for the raw medical data that are uploaded in Swarm. The index manager is developed such a way so that the contract keeps the metadata of metadata to the raw records which are only a single address (called swarm-hash) that points to an index file kept in Swarm.

Broadly, the index manager collects the Swarm-hashes of raw medical records in Swarm through the upload-manager and update those in an index file in a timely fashion. As mentioned earlier, Swarm is an immutable peer-to-peer storage, i.e., the file written once in Swarm cannot be updated in place. Hence, in this context update means creating a new version of the file with updated content (as shown in Fig. 8.4). Once the new version of the index file is created in Swarm, its address is

updated in the contract. Now, that the patient obtained the indexes of his medical records from the contract he can access it using bzz protocol (or its variation, such as bzzi and bzzr) from the Swarm-gateway. The file cannot be accessed without this swarm-hash.

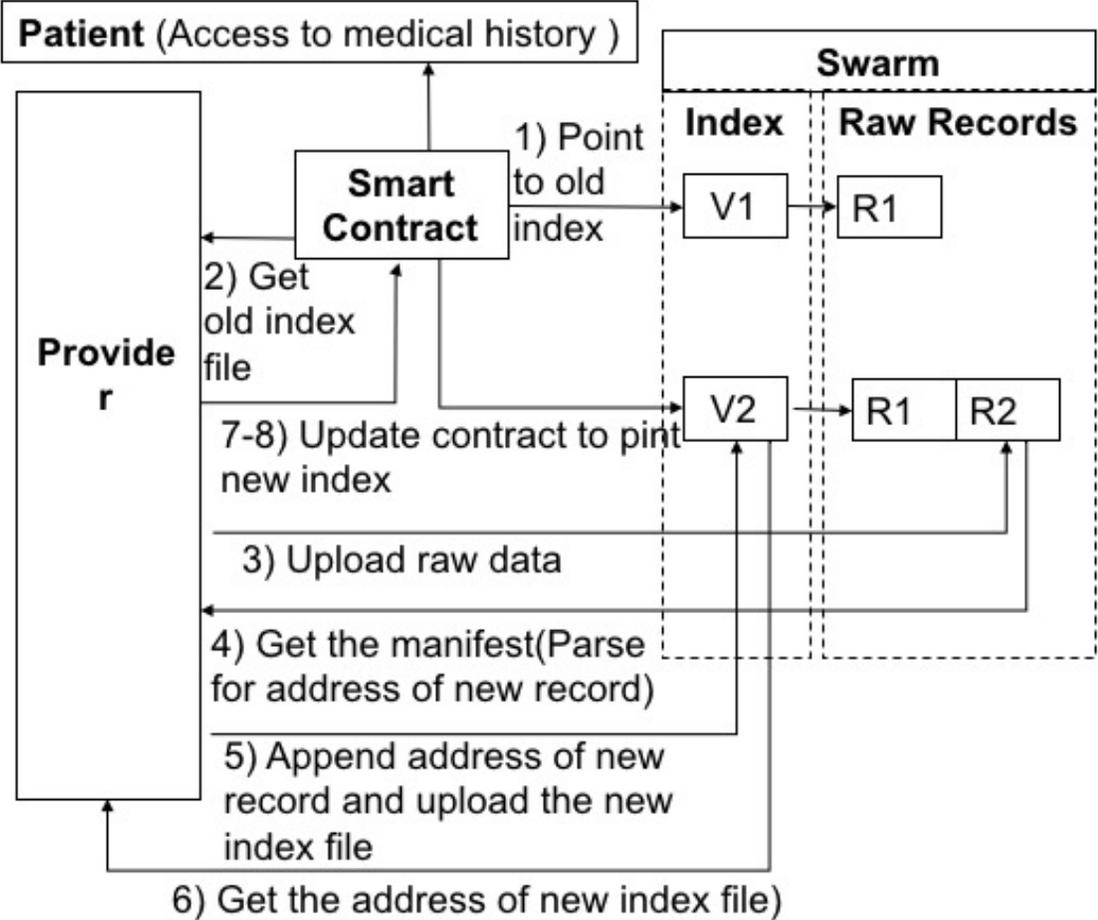


Figure 8.4: SwarMed indexing service

Figure 8.4 shows the update procedure for the immutable index file in Swarm and the contract in the Blockchain. Initially, the contract holds the pointer of an older version of the index file kept in Swarm. For a new patient registered in the system, the contract points to an empty index file. The provider gets an old list of swarm-hashes (or, an empty list for a new patient) by accessing the contract between the patient and the provider. Gradually, the providers start adding patient's medical records to swarm and populate the index file with the

corresponding swarm-hashes. Each time a provider upload the medical record to Swarm, Swarm returns the manifest of these data files (or, directory). SwarMed parses the manifest to get the swarm-hash value (i.e., the location) of the newly uploaded files (or directory). The provider appends this swarm-hash with the old list of swarm-hashes obtained from the old version of the contract and create a new index file. Finally, the new index file is uploaded to Swarm and notify the contract manager with the swarm-hash of this new index file.

It is to be noted, that the providers upload the raw medical record for their patient only once in Swarm (similar to the existing database system). Based upon the data-sharing-agreement defined in the contract an index file is created pointing to the required subset of records and the contract is updated with the address (swarm-hash) of the index file only. This way, SwarMed keeps the block-size small and constant and avoids any data duplication.

3. Contract Manager: It provides the interface between the index manager and the Ethereum contracts. It keeps track of all the patient's contract in the system and accesses the corresponding contract when required. Based on the contract, the contract manager is responsible for reading or updating the address stored in the contract (i.e., the address of the index file in Swarm), set or reset any permission flags (e.g., patient's consent on sharing a data). In sum, all the logistics that are instantiated with an Ethereum smart contract are executed by the contract manager.

8.5 Evaluation

The interoperability architecture of Swarmed has been evaluated mainly in terms of throughput and latency. The thesis evaluates each of its components separately and the architecture as a whole considering many design alternatives.

8.5.1 Dataset

For this chapter, a synthetic and anonymous dataset of 80million patient records are used following the PCORnet Common Data Model (CDM) which allows for the systematic analysis of disparate observational databases. As shown in Table 8.1 this work has identified 130 different attributes for a patient and simulated the records. Each records is 1.6KB in size. Like the real world, each patient is assumed to visit many providers many times in his life span.

Table 8.1: Common Data Model (CDM) of health care. All the records are synthesized with computer programs following this model

patid, birth_date, birth_time, sex, hispanic, race, biobank_flag, raw_sex, raw_hispanic, raw_race, conditionid, encounterid, report_date, resolve_date, onset_date, condition_status, condition_type, condition_source, raw_condition_status, raw_condition_type, raw_condition_source, birth_date, birth_time, biobank_flag, raw_sex, raw_hispanic, raw_race, diagnosisid, enc_type, admit_date, providerid, dx, dx_type, dx_source, pdx, raw_dx, raw_dx_type, raw_dx_source, raw_pdx, admit_date, admit_time, discharge_date, discharge_time, facility_location, enc_type, facilityid, discharge_disposition, discharge_status, drg, drg_type, admitting_source, raw_siteid, raw_enc_type, raw_discharge_disposition, raw_discharge_status, raw_drg_type, raw_admitting_source, lab_result_cm_id, lab_name, specimen_source, lab_loinc, priority, result_loc, lab_px, lab_px_type, lab_order_date, specimen_date, specimen_time, result_date, result_time, result_qual, result_num, result_modifier, result_unit, norm_range_low, norm_modifier_low, norm_range_high, norm_modifier_high, abn_ind, raw_lab_name, raw_lab_code, raw_panel, raw_result, raw_unit, raw_order_dept, raw_facility_code, prescribingid, encounterid, rx_providerid, rx_order_date, rx_order_time, rx_start_date, rx_end_date, rx_quantity, rx_refills, rx_days_supply, rx_frequency, rx_basis, rxnorm_cui, raw_rx_med_name, raw_rx_frequency, raw_rxnorm_cui, patid, proceduresid, enc_type, admit_date, px_date, px, px_type, px_source, raw_px, raw_px_type, vitalid, measure_date, measure_time, vital_source, ht, wt, diastolic, systolic, original_bmi, bp_position, smoking, tobacco, tobacco_type, raw_diastolic, raw_systolic, raw_bp_position, raw_smoking, raw_tobacco, raw_tobacco_type
--

8.5.2 Compute Environment

For the evaluation purpose the LSU HPC cluster called SuperMic is used. As shown in Table 8.2, each node has 2 Intel IveBridge Xeon processor with 10 cores each yielding a total of 20cores per node. Each node has 64GB of DRAM and one hard disk drive (HDD) attached. The throughput of the disk is evaluated to 160MB/s which means a total of 106667 records from the dataset can be written to the disk per second. To match the real world scenario, the 1Gpbs Ethernet interface is used for all the benchmark.

Table 8.2: Compute Environment

Total number of nodes	16
Processor/node	2 Intel IvyBridge
#cores/node	20
Storage/node	1 HDD
I/O Bandwidth/node	160MB/s
DRAM/node	64GB
Netwrok interface used	1Gbps Ethernet
Effective bandwidth (iperf)	941Mbps

8.5.3 Design Alternatives Evaluated

The following four design alternatives have been evaluated to show the relative merits of the architecture:

1. Traditional HTTP-based design: In this case, the patient’s medical data is stored in a standard HTTP server. It is the most commonly used infrastructure to transfer data over the Internet including the cloud-based architectures also.

2. P2P storage (Swarm and IPFS): The entire patient dataset is stored on the P2P storage and accessed via its hash-based guarantee of data integrity. The clients can join and leave the network any time they wish. Unlike HTTP, the data is replicated over multiple clients automatically when they join the network and is downloaded from the nearest source possible. Swarm and IPFS are evaluated individually to select the most sustainable architecture for the big data transfer.

3. Blockchain storage: This scenario uses the smart contract storage to store the patient records providing immutability and reliable time stamping on the data itself. Avoiding the need to manage a separate data store, this solution stores the data in smart contract permitting the checking of individual patient record.

5. Blockchain + HTTP-storage: In this case, the query string and the server address are stored in the smart-contract of Ethereum. On successful execution of the smart-contract, the data is fetched from the HTTP server in a traditional HTTP-based manner.

4. Blockchain + P2P-storage: This is similar to the previous one but instead of the HTTP-based query string or server address, the content-hash of the data is stored in the Ethereum smart-contract. In this design, all the hashes are stored in the Blockchain providing the immutability guarantee at the dataset level. The SwarMed architecture discussed earlier in 8.4 basically an enhancement over this design alternative.

8.5.4 Transferring Big Data over HTTP and P2P

To point out the scalability limitations of HTTP, an NPM version of the HTTP-server is setup in one of the nodes in SuperMic cluster and 8million patient records are kept inside the server. All the clients then read the data simultaneously from that server. To reduce the I/O bottleneck of the server, three replicas of the dataset has been made and the clients are scheduled in a round robin fashion to access the data. For P2P storage, both Swarm and IPFS are used both of which automatically replicate the data in different nodes of the clusters as each node work both as a client and a storage server.

Because of the content-based hash, all-server-all-client design and multiple replication of the dataset over P2P servers, P2P clients can read (download) the data from the nearest possible server without any bottleneck issue and is expected to scale uniformly. On the contrary, the HTTP clients need to download the data

from a particular server which is expected to show scalability issues with growing number of clients.

Figure 8.5 substantiates the claim. Although HTTP performs better for a small number of clients, its performance degrades almost linearly with the growing number of clients. Both the P2P storage, Swarm and IPFS on the other hand, show a uniform performance over growing number of clients (or, servers). Hence, the P2P storage is more sustainable and cost-effective in the healthcare scenario where millions of clients (e.g., patients, providers, and other third-party organizations) reads (download) data over the Internet every single day. In a production environment, the service provider’s operating cost increases linearly to scale the HTTP with growing number of clients as many servers need to be deployed.

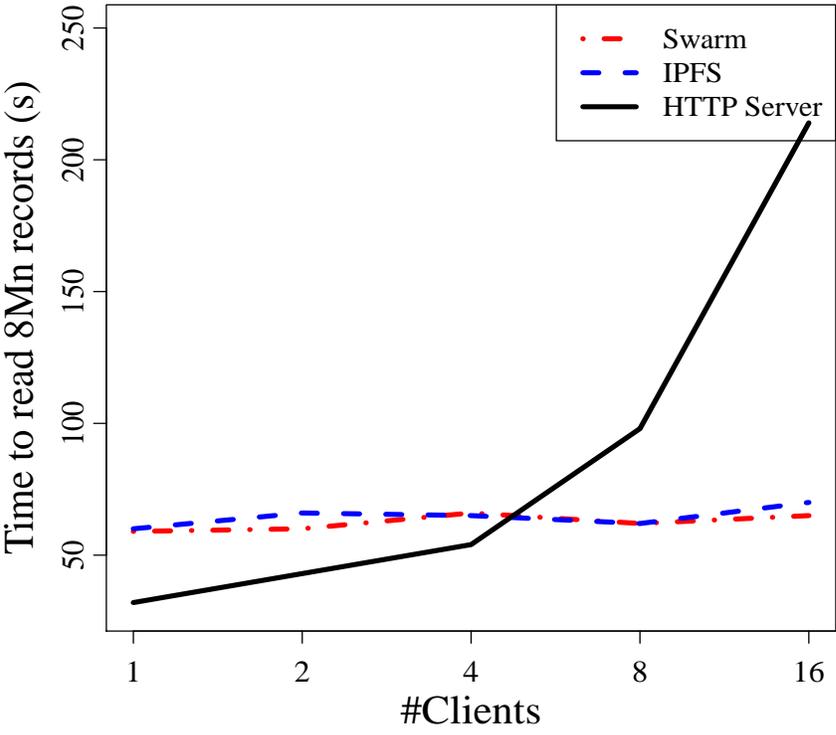


Figure 8.5: Transferring 8Mn patient record over P2P storage and from an HTTP server

8.5.5 Swarm vs IPFS

Figure 8.6 compares the I/O throughput of Swarm and IPFS in terms of strong scalability. That is one Swarm or IPFS node writes 8million unique patient record to a cluster of varying size. Swarm shows more than 2x performance gain comparing to IPFS. Furthermore, a slight increase is observed in the execution time of IPFS with the increase in the number of nodes whereas Swarm shows similar performance.

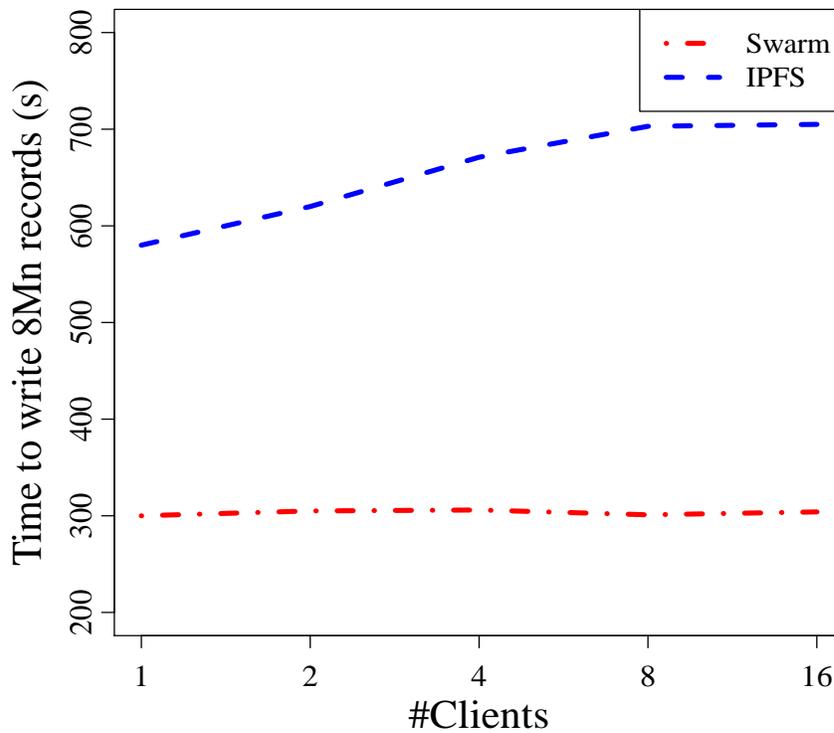
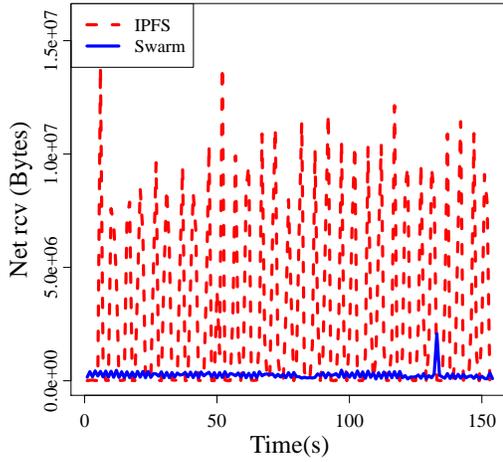


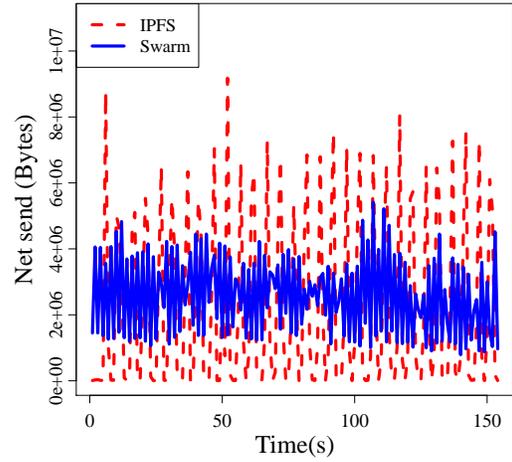
Figure 8.6: Writing 8Mn patient records

To find the root cause of the behavior, the I/O pattern of both network and the local file system for both Swarm and IPFS is observed on their respective writer node (the node which writes/uploads the data from local file system to the P2P cluster). The system characteristics are shown in Figure 8.7.

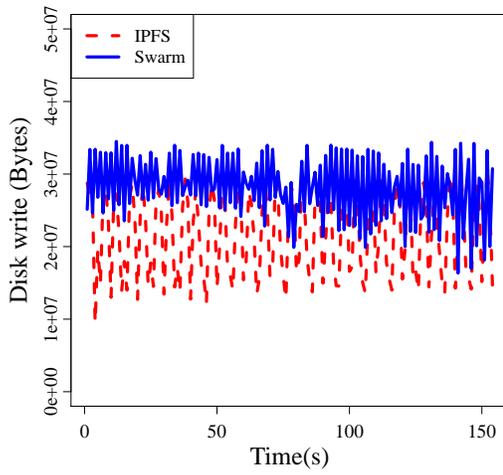
First, significantly less incoming traffic was observed in the writer node of



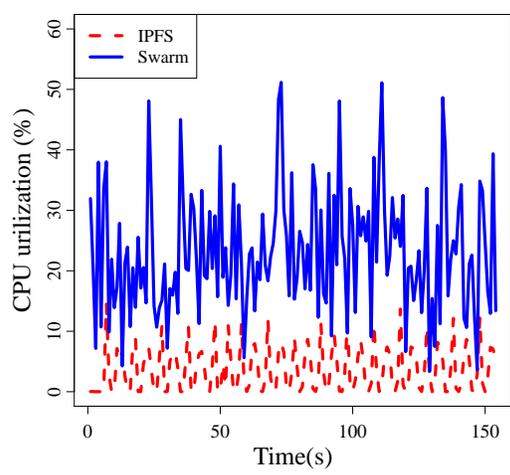
(a) Network received byte statistics



(b) Network sent byte statistics



(c) Disk write statistics



(d) CPU utilization statistics

Figure 8.7: Swarm and IPFS writer node statistics

Swarm comparing to that of IPFS (Figure 8.7a) concluding that Swarm uses more efficient messaging service and relatively lower size of the message to synchronize its peers comparing to IPFS. Then to delve details into the writing strategy of swarm and IPFS, the outgoing network traffic (Figure 8.7b) was observed and the disk I/O pattern (Figure 8.7c). Swarm uses a lazy approach for write synchronization and replication where it writes the data onto the local disk first and then sends to the peers in small packets. On the contrary, IPFS writes a smaller amount of data to disk and sends it over the network for replication on other peers resulting in significantly higher network traffic. Consequently, Swarm shows significantly better CPU utilization (Figure 8.7d) comparing to that of IPFS yielding better write performance.

However, Swarm and IPFS both performed similarly for reading (download) in the cluster environment used in this chapter. This is because of their similar key-based routing algorithm based on xor logarithmic distance.

A typical healthcare interoperability network is both read and write heavy. Thousands of patients visit hundreds of different providers' site generating terabytes of data that need to be written (uploaded) and be available to the patients and other third-party stakeholders to read (download). It is different from traditional write-once-read-many applications. Hence the performance should not be bottlenecked by the read or write performance of the storage platform making Swarm obviously a better choice in the application.

8.5.6 Blockchain Performance

Although Blockchain provides a solution to the challenges involved in providing trust-less service in a secured and privacy-preserved way, its wide-scale adoption is still hindered by its longer service-time and throughput especially when big data is involved. Figure 8.8 shows a linear loss (First 4 groups of the Figure 8.8) in Ethereum Blockchain's performance with increasing size of data. Although

the average size of each record is only 1.6KB in the experiments, Ethereum’s transactional throughput shows a sharp decline with increasing number of patients’ records per transaction even-though there are less than 10 records (i.e. 16KB only) per transaction. In terms of record-throughput, i.e., the number patient’s records can be written and transferred through the main chain per second also decreases. The similar trend can be found during reading the data also.

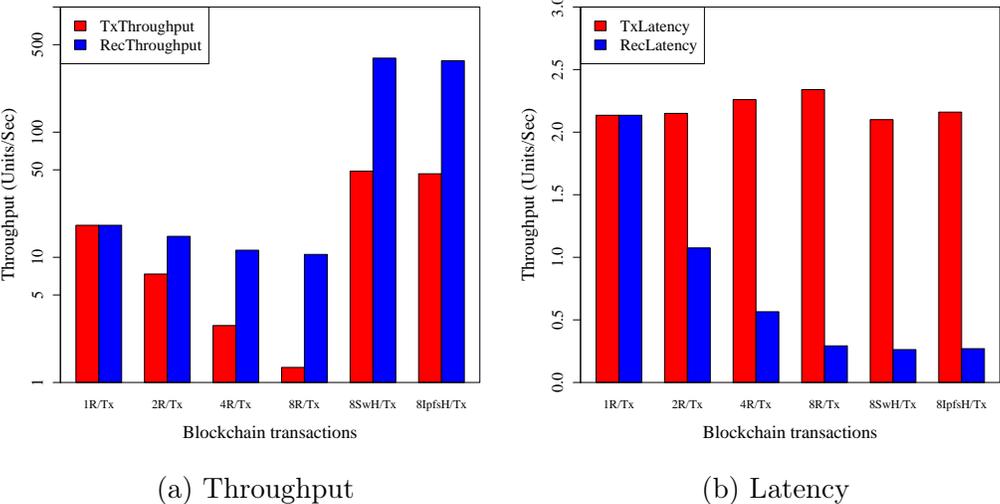


Figure 8.8: Blockchain performance

However, the transactional latency (first 4 groups of Figure 8.8b) remains almost same yielding an improvement in the record latency that is the total time spent to write and transfer one record through the chain.

Hence, a significant trade-off between throughput and latency is observed while using the Blockchain’s storage (i.e., saving the data directly on the smart contract) for a big dataset. On the contrary, the healthcare applications demand both high throughput and low latency. For example, When multiple patient’s records are shared with the scientists the system demands the throughput whereas, in a heavily loaded emergency, the data should be written on the system as quickly as possible demanding low latency.

8.5.7 Blockchain + HTTP

Many of the existing dApp in different domains including healthcare follow this design nowadays. Although the design apparently looks efficient it has two fundamental flaws as follows:

1) For a small number of clients, the design may guarantee efficient data transfer with HTTP along with the security promises of Blockchain. However, the first one shows scalability issue (Figure 8.5) with growing clients whereas the second one shows tremendous bottleneck with growing size of data.

2) As the data is stored in the database or providers' site, the design does neither provide any guaranty on preventing DDoS nor it provides any tamper resistance mechanism. That is, the design consideration does not provide the fundamental requirement desired in the domain of healthcare (and possibly other domain also). Consequently, this design alternative is not evaluated in this work.

8.5.8 Blockchain + Swarm

As discussed earlier, in this design consideration, the data resides inside a P2P storage and the content-hash (64Bytes) is stored in the Blockchain transaction. The system is tamper-proof, free from DDoS attack and at the same time offload the data from the main chain to guarantee the throughput.

The last two bars of Figure 8.8 compares the performance of this design to the Blockchain-only design. As the data size on the transaction decreases the gas used per transaction also decreases in this current design. Consequently, many transactions are accumulated in a block and mined simultaneously. Hence, in a busy time when many transactions enter the Blockchain system from many clients, the throughput will be significantly more. As shown in Figure 8.8a this design alternative produce 25x better throughput in terms of the number of Blockchain transaction per second compared to Blockchain alone. In terms of the number of records transferred per second, the corresponding gain is 21x.

Moreover, this design improves both the throughput and latency of the system (Figure 8.8a and 8.8b) unlike the Blockchain-only design which poses a significant trade-off between these two.

It should also be observed that in conjunction with Blockchain, both Swarm and IPFS performs similarly although Swarm showed better performance over IPFS.

The major reason for selecting Swarm over IPFS in the design is its better sustainability on big data system. Although a large-size file is not very common in healthcare domain and it is flooded with millions of small-size file, as soon as the file size grows beyond 10GB (e.g., a genomic sequence of a patient), the file read/write time starts dominating the throughput of the system and Swarm shows better promises at that time. Since one of the motivations is to transfer large-scale anonymous data between medical school and the Universities to foster translational research Swarm is chosen over IPFS. However, the implementation of consent management can easily be decoupled from the storage and can easily migrate to IPFS once the limitations are amended.

8.5.9 Scalability of Blockchain with Swarm:

Figure 8.9 and 8.10 shows the scalability of the design. Only one swarm hash is stored per Blockchain transaction pointing to a single patient record kept in its P2P storage. To simulate the busy scenario multiple clients have been assigned to each Ethereum node where each of the clients sends multiple transaction requests at once. That is, at a single time slot (less than a second) the total number of transactions in the system can be given by $\#clients \times \#requests \times \#nodes$. In these set of experiments $\#requests$ is set to 2000. When there are 16 nodes, 16 clients are assigned to each node there are $16 \times 2000 \times 16 = 512000$ transaction requests are in the system. As it can be seen in Figure 8.9a and 8.10a, in a busy scenario with multiple clients and thousands of requests the design is weekly scalable for

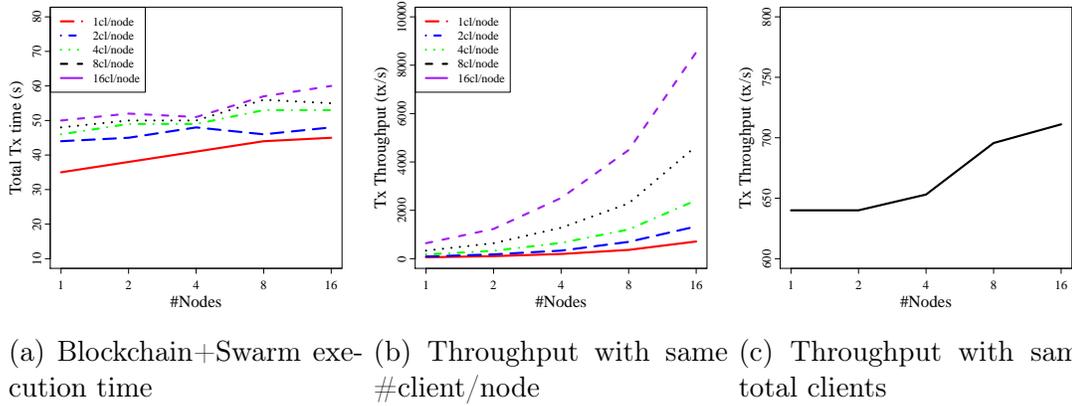


Figure 8.9: Blockchain+Swarm write scalability

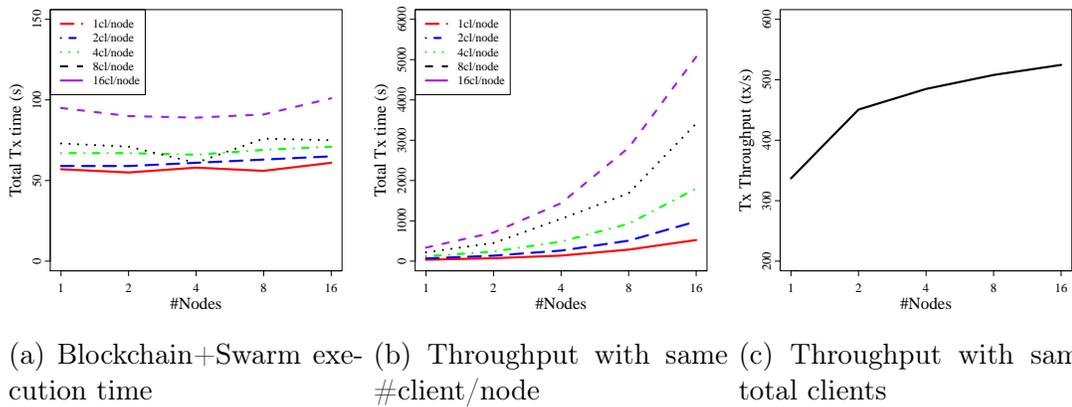


Figure 8.10: Blockchain+Swarm write scalability

both read and write operations (i.e., upload and download operations). That is, the execution time remains almost similar with increasing number of Ethereum nodes when the number of clients (alternatively number of transaction requests) per node also increase at the same proportion. Each Swarm hash points to only one patient record in this set of experiments yielding the same record throughput per transaction.

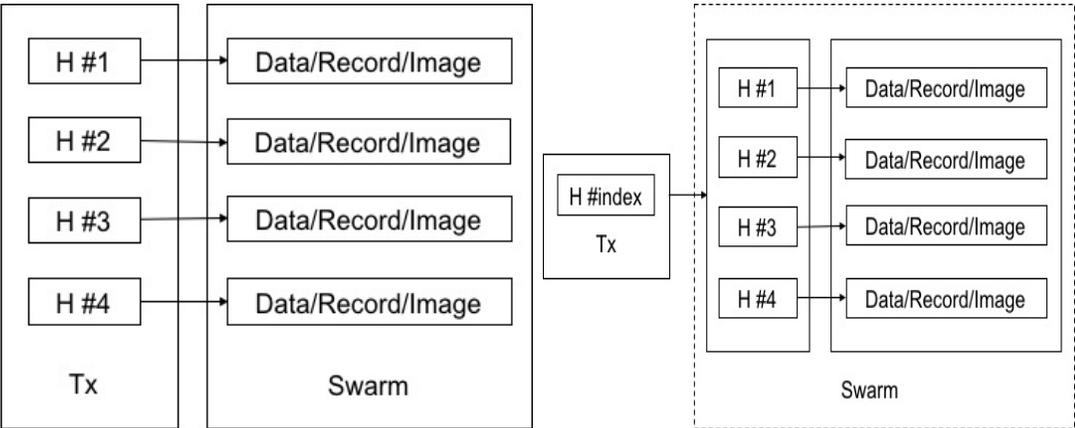
A direct interpretation of this result shows the throughput of the system when keeping the number of the client same per node. As it can be seen in Figure 8.9b and 8.10b, the total number of transactions executed per second improves almost linearly with the increase in number of nodes and the. On the other hand Figure

8.9c and 8.10c shows the throughput of the design when keeping the total number of client same (16 in the experiment) in the system.

Hence, Blockchain with Swarm shows significant performance gain and scalable behavior with both increase in the number of clients and nodes

8.5.10 Blockchain + Swarm + Indexing

Although Swarm in conjunction with Blockchain shows significantly better performance comparing to the Blockchain alone, the performance of the system can be bottlenecked with the number of hashes stored in a transaction as shown in Figure 8.11.



(a) Design alternative #1: Keep all the Swarm Hashes on Tx (b) Design alternative #2: Keep only the Index Swarm Hash on Tx

Figure 8.11: Blockchain+Swarm design alternatives

To evaluate the benefit of the indexing, a 25MB file is created including 16700 records per file. Although these many records per patient are not common in the real world, the byte size reflects the presence of x-ray images, mammogram images, etc. For each of these files, a Swarm hash is written on the transaction. Since the major bottleneck is observed in the data size in Blockchain transaction and not in the Swarm, the experiment reflects the real world scenario giving a good quantitative metric to express the capability of the system.

To pinpoint the benefit of the design, one client is assigned per node of a 16

node Ethereum cluster each working as a Swarm peer also. Each client sends 200 transaction requests. The total amount of data migrated through the system can be given by $\#nodes \times \#clients/node \times \#requests/client \times swarmFileSize$. That is, for a 16node cluster, 1 client per node with 200 requests per clients, a total of 80GB ($16 \times 1 \times 200 \times 25MB$) data is migrated to the peers.

The first design alternative has two phases executed sequentially such as, 1) write to the swarm and 2) write n -th hash to the smart contract. Whereas, the second design alternative has four different phases such as, 1) write data to swarm, 2) read index-hash from Blockchain 3) read index-content from swarm 4) add index-content to swarm with new data-hash 5) add the new index-hash to the Blockchain

Although the second alternative has many steps involved in it, most of them can execute in a constant time and the cumulative time for all these steps are significantly less comparing to the first step of the first design alternative when $n > 1$.

Figure 8.12 compares both the design. As it can be seen, the average execution time of the first design alternative i.e., many swarm hashes on the transaction increases exponentially with increase in the number of swarm hashes. On the other hand, the index-based design performs almost similar for any number of Swarm hashes as the Blockchain is kept lightweight always.

The record-throughput of both the system can be calculated as $(\#records/client \times \#clients/node \times \#nodes) / averageExecutionTime$. For a 16 node cluster, 1 client per node, 16700 records written by each client, the fist alternative shows a throughput of 31010.60 ($16700 * 1 * 16 * 64 / 551.45$) records/s where as the proposed index-based design shows 9x performance gain yielding a throughput of 267033.10 ($16700 * 1 * 16 * 64 / 64.04$) records/s

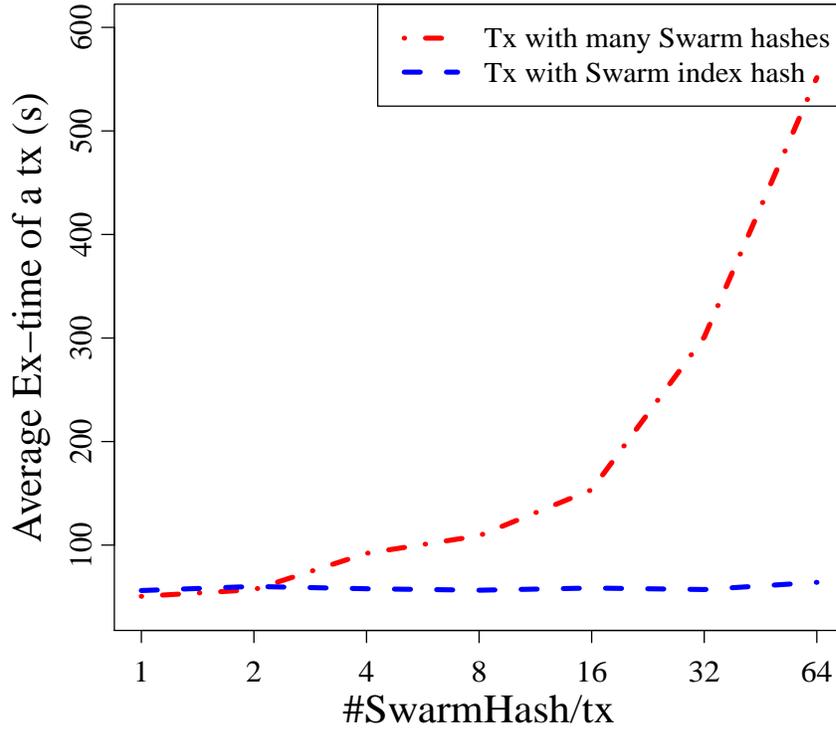


Figure 8.12: Execution time for upload 8Mn patient records to Swarm for sharing

8.6 Operating Cost of SwarMed on Public Network (Internet)

Table 8.3 shows the total operating cost of storage for different design alternatives. It can be easily observed that the proposed design of SwarMed outperforms all the other alternatives in terms of cost. For a fair comparison, we calculated the cost of 1GB of patient records over the span of 10years which is a common scenario in the domain.

Table 8.3: Operating cost of different designs

Storage type	Data size	Time of storage	Total cost (USD)
Ethereum + Swarm + Index (Proposed)	1GB	Immutable	0.34
Ethereum only	1GB	Immutable	80500000
Amazon EBS (HTTP)	1GB	10 year	12.00
Azure RA-GRS (HTTP)	1GB	10 year	14.40

The cost of the Ethereum-based design architectures are calculated using Gas, the unit of measuring the computation work or storage in Ethereum. As mentioned in the Ethereum yellow paper [17], the fee to store 256bit word is 20K Gas. The standard Gas-price is 3GWei (3×10^{-7} Eth)¹. Assuming the value of 1Eth as 470.61USD², the fees for storing the 256bit word can be calculated as 0.006Eth i.e., 2.72USD. Hence, the cost of storing one bit is 0.010625USD. Using this information, the cost of each of the design shown in Table 8.3 can be calculated as follows:

1) The proposed design of Swarmed stores only one Swarm hash of size 32bit on the chain of Ethereum. Using the above information the total storage cost can be calculated as 0.34USD (32×0.010625).

2) If the entire 1GB of data is stored on the chain of Ethereum, the total cost of storage can be calculated as 80500000USD ($8 \times 10^{10} \times 0.010625$).

3) The last two alternatives show the cost of storing 1GB of data in two popular cloud-based storage, Amazon-EBS and Azure-RA-GRS. For a fair comparison with immutable storage of Ethereum or Swarm, we calculated the cost of 1GB storage in these cloud-based platform over 10years. Using the cost information available in the corresponding websites^{3, 4}, it is observed that the existing cloud-based storages charges almost 35 to 42 times more per GB of storage compared to that of the proposed design of SwarMed over a 10years of time span.

8.7 SwarMed and ONC's Interoperability Roadmap

This section evaluates SwarMed in the context of ONC's interoperability roadmap [109] published in 2015.

By giving patients a immutable, trusted log of their medical history, the SwarMed system like MedRec[110], directly addresses the ONC interoperability

¹<https://ethgasstation.info/>

²<https://www.coindesk.com/ethereum-price/>

³<https://aws.amazon.com/ebs/pricing/>

⁴<https://azure.microsoft.com/en-us/pricing/details/storage/blobs/>

roadmaps principal outcome, Individuals have access to longitudinal electronic health information, can contribute to the information, and can direct it to any electronic location [109]

The current state of healthcare records is disjointed and fractured due to a lack of common architectures and standards that would allow the safe transfer of sensitive information among stakeholders in the system. On the other hand, SwarMed provides a uniform, tamper-resistant, peer-to-peer storage for the electronic health information without any data duplication across nationwide systems (consisting of multiple providers, and stakeholders). Hence, the entire health information is consistent with authorization and access permission. Consequently, SwarMed is able to address the ONC's requirement for "secure and trusted exchange of electronic health information, consistent with privacy protections and individual's preferences, across states, networks, and entities". [109]

Furthermore, SwarMed enables big data sharing in the biomedical domain using Blockchain. The high throughput architecture with its security and privacy promises will drive the translational research including precision medicine, predictive analysis, etc. which depend largely on the availability of big dataset.

8.8 Conclusion

This chapter proposed SwarMed, a scalable and robust proof-of-concept to share large-scale medical data over the Ethereum Blockchain ecosystem. SwarMed addresses the industry's interoperability challenges by using block chain's decentralized security promises. At the same time, this work developed an efficient way of accessing data off-the-chain using Swarm in conjunction with the proposed indexing mechanism and surpasses the current throughput limitation of Blockchain by several magnitudes.

Deviating from the traditional database and centralized server, many application started using the p2p storage infrastructure for automatic scalability and

significantly lower cost overhead. A NoSQL-like query engine on top of this p2p storage solution can improve the efficiency of these applications by several magnitudes. From that direction also, SwarMed provide a good initial starting point.

The future research direction includes adding more feature-sets in the smart contracts to align the proof-of-concept fully with the existing regulations such as HIPAA, CURE, COPAA, etc. The engineering infrastructure should also be improved in future for better throughput and latency. A fully functional database service over Swarm can improve this aspect. The proposed indexing service provides a good initial starting point for that.

Chapter 9

Conclusion and Future Work

Big data is ubiquitous. Starting from genomic analysis to medical informatics, from astronomy to quantum physics, scientific applications are flooded with huge amounts of data today. The data intensive nature of the applications are rapidly shifting the computational and architectural model of traditional HPC at different levels including the software programming model, cyberinfrastructure and transaction over network. Scalability became the most desired characteristics at all these while at the same time the costly bandwidth needs to be preserved.

This thesis first addresses the issue of scalability by designing novel algorithms and proposing novel software frameworks for different scientific applications. Unlike traditional MPI- or grid-based algorithms, the proposed algorithms and frameworks are locality-based. That is, instead of moving the large datasets towards the small-size computation these algorithms move the small-size computation to the large datasets. This strategy saves the costly bandwidth. To this end, the thesis focuses mainly on the large-scale genome analysis pipeline which recently has made its way to the forefront of big data challenges. The algorithms developed in this thesis to handle these big data are appreciably accurate compared to the existing tools and can scale over a hundreds of compute nodes with terabytes of data. Furthermore, by using lower number of cores and memory per node, the thesis showed that these algorithms and software frameworks can run on top of scaled out cluster of commodity hardware. By developing scientific applications for scaled out cluster, the thesis in one hand addresses the scalability issues in the existing applications. On the other hand, by eliminating the need for sophisticated HPC hardware, the thesis addresses the cost issues involved in data driven science. One of our future research direction is generalize the proposed frameworks and

algorithms to address several other applications in the large-scale genomic analysis pipeline such as variant calling, metagenomic analysis, gene finding etc.

The characteristics of the algorithms and software framework developed in the thesis are substantially new. Consequently, there is limited understanding of the underlying cyber infrastructure that they need for good performance. Hence, the thesis also evaluates a broad range of cluster architecture required for the good performance of these software applications. The rapid development in storage and processor architecture have already changed the performance point. In collaboration with Samsung Ltd., S. Korea, the thesis evaluates the performance implication of different cutting edge hardware such as SSD, NVMe SSDs, etc. It also provides significant insight on how to deploy a high performance big data analytic cluster for data intensive science. The thesis also identified the need for a balanced cluster in terms of both performance and economy. In the last decade HPC providers such as NSFCloud and XSEDE have invested millions of dollars to provide resources for data driven scientific applications with the notion that 'big data needs big resources'. At this inflection point of HPC infrastructure the thesis addresses the need for a balanced cluster architecture by providing a theoretical model for optimal cluster architecture. Instead of increasing the FLOPS only, the HPC cluster for data science should be balanced in terms of FLOPS, I/O bandwidth and DRAM. The system designer needs to consider more degrees of freedom and answer the questions such as 'how much memory or I/O bandwidth required per FLOPs'. By answering such questions, the model provides an easy-to-use guideline for setting up an HPC cluster for big data analytics. The designers can make informed choice of hardware components to deploy a scalable and cost-effective cluster for scientific big data analysis even when the application characteristics is not known.

Finally, the thesis provides a high throughput yet cost-effective solutions to transfer huge amount of big data in different geographic locations. Deviating from

traditional client-server architecture and centralized data transfer model the thesis proposes a decentralized interoperability model to share and transfer large volume of data in a secured, privacy-protected and tamper-proof fashion. The proposed data transfer architecture improves the throughput of the current Blockchain-based decentralized transactions by several magnitudes. An indexing mechanism over a P2P storage model is proposed so that the architecture can scale with increasing amount of data. The decentralized interoperability model is automatically scalable with increasing number of clients as all the clients work as a storage-server also to facilitate improved and high throughput data transfer over geographically separated locations. By making each client a storage server, the decentralized architecture also reduces the operating cost significantly comparing to traditional HTTP or FTP-based server where more servers are required to facilitate increasing number of clients. This part uses a large synthetic biomedical dataset.

References

- [1] A. K. Das, S. Goswami, and R. Platania, "Ibm power8 hpc system accelerates genomics analysis with smt8 multithreading," http://www.lsu.edu/mediacenter/docs/LSU-IBM_POWER8_GenomeBenchmark.pdf.
- [2] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.
- [3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, 2008.
- [4] T. Cheatham, A. Fahmy, D. Stefanescu, and L. Valiant, "Bulk synchronous parallel computing – a paradigm for transportable software," in *Tools and Environments for Parallel and Distributed Systems*. Springer, 1996, pp. 61–76.
- [5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.
- [6] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.
- [7] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, 2011.
- [8] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *Proceedings of OSDI*, 2014, pp. 599–613.
- [9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.
- [10] D. Rangel, "Dynamodb: Everything you need to know about amazon web service's nosql database," 2015.
- [11] J. Lehnardt and N. Slater, "Couchdb: The definitive guide. time to relax," 2009.
- [12] M. Johns, *Getting Started with Hazelcast*. Packt Publishing Ltd, 2015.
- [13] J. L. Carlson, *Redis in Action*. Manning Publications Co., 2013.

- [14] A. H. Team, "Apache hbase reference guide," *Apache, version*, vol. 2, no. 0, 2015.
- [15] K. Chodorow, *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.
- [16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [17] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.
- [18] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.
- [19] V. Tron and A. Fischer. (2016) Swarm serverless hosting incentivised peer-to-peer storage and content distribution. [Online]. Available: <http://swarm-gateways.net/bzz:/theswarm.eth/>
- [20] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.
- [21] A. K. Das, S. Shams, S. Goswami, R. Platania, K. Lee, and s.-J. Park, "Parsech: Parallel sequencing error correction with hadoop for large-scale genome," in *Proceedings of the 9th International BICob Conference*. ISCA, 2017.
- [22] D. I. Lou, J. A. Hussmann, R. M. McBee, A. Acevedo, R. Andino, W. H. Press, and S. L. Sawyer, "High-throughput dna sequencing errors are reduced by orders of magnitude using circle sequencing," *Proceedings of the National Academy of Sciences*, vol. 110, no. 49, 2013.
- [23] Y. Gu, Q. Zhu, X. Liu, Y. Dong, C. T. Brown, and S. Pramanik, "Using disk based index and box queries for genome sequencing error correction."
- [24] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, "Quake: quality-aware detection and correction of sequencing errors," *Genome biology*, 2010.
- [25] W.-C. Kao, A. H. Chan, and Y. S. Song, "Echo: a reference-free short-read error correction algorithm," *Genome research*, vol. 21, no. 7, 2011.
- [26] X. Yang, K. S. Dorman, and S. Aluru, "Reptile: representative tiling for short read error correction," *Bioinformatics*, vol. 26, no. 20, 2010.
- [27] S. Saha and S. Rajasekaran, "Ec: an efficient error correction algorithm for short reads," *BMC bioinformatics*, vol. 16, no. 17, p. 1, 2015.
- [28] C. T. Brown, A. Howe, Q. Zhang, A. B. Pyrkosz, and T. H. Brom, "A reference-free algorithm for computational normalization of shotgun sequencing data," *arXiv preprint arXiv:1203.4802*, 2012.

- [29] P. A. Pevzner, H. Tang, and M. S. Waterman, “An eulerian path approach to dna fragment assembly,” *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, 2001.
- [30] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner, “Error correction of high-throughput sequencing datasets with non-uniform coverage,” *Bioinformatics*, vol. 27, no. 13, 2011.
- [31] L. Ilie and M. Molnar, “Racer: Rapid and accurate correction of errors in reads,” *Bioinformatics*, 2013.
- [32] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, “Shrec: a short-read error correction method,” *Bioinformatics*, vol. 25, 2009.
- [33] L. Ilie, F. Fazayeli, and S. Ilie, “Hitec: accurate error correction in high-throughput sequencing data,” *Bioinformatics*, vol. 27, no. 3, 2011.
- [34] L. Song, L. Florea, and B. Langmead, “Lighter: fast and memory-efficient sequencing error correction without counting,” *Genome biology*, vol. 15, no. 11, 2014.
- [35] Y. Liu, J. Schröder, and B. Schmidt, “Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data,” *Bioinformatics*, vol. 29, no. 3, 2013.
- [36] Y. Liu, B. Schmidt, and D. L. Maskell, “Decgpu: distributed error correction on massively parallel graphics processing units using cuda and mpi,” *BMC bioinformatics*, vol. 12, no. 1, 2011.
- [37] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, “A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces,” *ACM Transactions on Information Systems (TOIS)*, vol. 24, no. 1, 2006.
- [38] X. Yang, S. P. Chockalingam, and S. Aluru, “A survey of error-correction methods for next-generation sequencing,” *Briefings in bioinformatics*, vol. 14, no. 1, 2013.
- [39] A. K. Das, P. K. Koppa, S. Goswami, R. Platania, and S.-J. Park, “Large-scale parallel genome assembler over cloud computing environment,” *Journal of Bioinformatics and Computational Biology*, 2017.
- [40] C. Avery, “Giraph: Large-scale graph processing infrastructure on hadoop,” *Proceedings of the Hadoop Summit. Santa Clara*, 2011.
- [41] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin: a not-so-foreign language for data processing,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1099–1110.

- [42] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for snps with cloud computing," *Genome Biol*, vol. 10, no. 11, p. R134, 2009.
- [43] H. Nordberg, K. Bhatia, K. Wang, and Z. Wang, "Biopig: a hadoop-based analytic toolkit for large-scale sequence data," *Bioinformatics*, p. btt528, 2013.
- [44] M. Schatz, D. Sommer, D. Kelley, and M. Pop, "Contrail: Assembly of large genomes using cloud computing," in *CSHL Biology of Genomes Conference*, 2010.
- [45] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: a parallel assembler for short read sequence data," *Genome research*, vol. 19, no. 6, pp. 1117–1123, 2009.
- [46] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, vol. 18, no. 5, pp. 821–829, 2008.
- [47] R. Chikhi and G. Rizk, "Space-efficient and exact de bruijn graph representation based on a bloom filter," *Algorithms for Molecular Biology*, vol. 8, no. 1, p. 22, 2013.
- [48] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "Allpaths: de novo assembly of whole-genome shotgun microreads," *Genome research*, vol. 18, no. 5, pp. 810–820, 2008.
- [49] R. Kajitani, K. Toshimoto, H. Noguchi, A. Toyoda, Y. Ogura, M. Okuno, M. Yabana, M. Harada, E. Nagayasu, H. Maruyama *et al.*, "Efficient de novo assembly of highly heterozygous genomes from whole-genome shotgun short reads," *Genome research*, vol. 24, no. 8, pp. 1384–1395, 2014.
- [50] Y. Liu, B. Schmidt, and D. L. Maskell, "Parallelized short read assembly of large genomes using de bruijn graphs," *BMC bioinformatics*, vol. 12, no. 1, p. 354, 2011.
- [51] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies," *Journal of Computational Biology*, vol. 17, no. 11, pp. 1519–1533, 2010.
- [52] J. A. Chapman, I. Ho, S. Sunkara, S. Luo, G. P. Schroth, and D. S. Rokhsar, "Meraculous: de novo genome assembly with short paired-end reads," *PloS one*, vol. 6, no. 8, p. e23501, 2011.
- [53] T. White, *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

- [54] A. V. Gerbessiotis and L. G. Valiant, “Direct bulk-synchronous parallel algorithms,” *Journal of parallel and distributed computing*, vol. 22, no. 2, pp. 251–267, 1994.
- [55] R. J. Anderson and G. L. Miller, “A simple randomized parallel algorithm for list-ranking,” *Information Processing Letters*, vol. 33, no. 5, pp. 269–273, 1990.
- [56] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, “Quast: quality assessment tool for genome assemblies,” *Bioinformatics*, vol. 29, no. 8, pp. 1072–1075, 2013.
- [57] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu, “A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers,” *BMC genomics*, vol. 13, no. 1, p. 341, 2012.
- [58] L. Salmela and E. Rivals, “Lordec: accurate and efficient long read error correction,” *Bioinformatics*, vol. 30, no. 24, pp. 3506–3514, 2014.
- [59] G. Miclotte, M. Heydari, P. Demeester, P. Audenaert, and J. Fostier, “Jabba: Hybrid error correction for long sequencing reads using maximal exact matches,” in *International Workshop on Algorithms in Bioinformatics*. Springer, 2015, pp. 175–188.
- [60] L. Salmela and J. Schröder, “Correcting errors in short reads by multiple alignments,” *Bioinformatics*, vol. 27, no. 11, 2011.
- [61] L. Salmela, R. Walve, E. Rivals, and E. Ukkonen, “Accurate self-correction of errors in long reads using de bruijn graphs,” *Bioinformatics*, vol. 33, no. 6, pp. 799–806, 2016.
- [62] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, “Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation,” *Genome research*, vol. 27, no. 5, pp. 722–736, 2017.
- [63] E. Haghshenas, F. Hach, S. C. Sahinalp, and C. Chauve, “Colormap: Correcting long reads by mapping short reads,” *Bioinformatics*, vol. 32, no. 17, pp. i545–i551, 2016.
- [64] S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis *et al.*, “Hybrid error correction and de novo assembly of single-molecule sequencing reads,” *Nature biotechnology*, vol. 30, no. 7, pp. 693–700, 2012.
- [65] K. F. Au, J. G. Underwood, L. Lee, and W. H. Wong, “Improving pacbio long read accuracy by short read alignment,” *PloS one*, vol. 7, no. 10, p. e46679, 2012.

- [66] T. Hackl, R. Hedrich, J. Schultz, and F. Förster, “proovread: large-scale high-accuracy pacbio correction through iterative short read consensus,” *Bioinformatics*, vol. 30, no. 21, pp. 3004–3011, 2014.
- [67] E. Bao and L. Lan, “Halc: High throughput algorithm for long read error correction,” *BMC bioinformatics*, vol. 18, no. 1, p. 204, 2017.
- [68] F. Y. Chin, H. C. Leung, W.-L. Li, and S.-M. Yiu, “Finding optimal threshold for correction error reads in dna assembling,” *BMC bioinformatics*, vol. 10, no. 1, p. S15, 2009.
- [69] M. J. Chaisson and G. Tesler, “Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory,” *BMC bioinformatics*, vol. 13, no. 1, p. 238, 2012.
- [70] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows-wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.
- [71] A. K. Das, S.-J. Park, J. Hong, and W. Chang, “Evaluating different distributed-cyber-infrastructure for data and compute intensive scientific application,” in *IEEE International Conference on Big Data*, 2015.
- [72] E. Georganas, A. Buluç, J. Chapman, L. Olikar, D. Rokhsar, and K. Yelick, “Parallel de bruijn graph construction and traversal for de novo genome assembly,” in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 2014, pp. 437–448.
- [73] Y. Li, P. Kamousi, F. Han, S. Yang, X. Yan, and S. Suri, “Memory efficient minimum substring partitioning,” in *Proceedings of the VLDB Endowment*, vol. 6, no. 3. VLDB Endowment, 2013, pp. 169–180.
- [74] Z. Fadika, M. Govindaraju, R. Canon, and L. Ramakrishnan, “Evaluating hadoop for data-intensive scientific operations,” in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 67–74.
- [75] S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox, “A tale of two data-intensive paradigms: Applications, abstractions, and architectures,” in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 645–652.
- [76] U. C. Satish, P. Kondikoppa, S. Park, M. Patil, and R. Shah, “Mapreduce based parallel suffix tree construction for human genome,” in *20th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2014, Hsinchu, Taiwan, December 16-19, 2014*, 2014, pp. 664–670.

- [77] P. Kondikoppa, C.-H. Chiu, C. Cui, L. Xue, and S.-J. Park, “Network-aware scheduling of mapreduce framework on distributed clusters over high speed networks,” in *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*. ACM, 2012, pp. 39–44.
- [78] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni, and D. K. Panda, “Performance analysis and evaluation of infiniband fdr and 40gige roce on hpc and cloud computing systems,” in *High-Performance Interconnects (HOTI), 2012 IEEE 20th Annual Symposium on*. IEEE, 2012, pp. 48–55.
- [79] J. Yu, G. Liu, W. Hu, W. Dong, and W. Zhang, “Mechanisms of optimizing mapreduce framework on high performance computer,” in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 708–713.
- [80] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park, “Enabling cost-effective data processing with smart ssd,” in *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2013.
- [81] D. Wu, W. Luo, W. Xie, X. Ji, J. He, and D. Wu, “Understanding the impacts of solid-state storage on the hadoop performance,” in *International Conference on Advanced Cloud and Big Data*, 2013.
- [82] S. Moon, J. Lee, and Y. S. Kee, “Introducing ssds to the hadoop mapreduce framework,” in *IEEE 7th International Conference on Cloud Computing (CLOUD)*. IEEE, 2014, pp. 272–279.
- [83] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, “A platform for scalable one-pass analytics using mapreduce,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 985–996.
- [84] K. Krish, A. Khasyanski, G. Wang, A. R. Butt, and G. Makkar, “On the use of shared storage in shared-nothing environments,” in *IEEE International Conference on Big Data*. IEEE, 2013, pp. 313–318.
- [85] W. Tan, L. Fong, and Y. Liu, “Effectiveness assessment of solid-state drive used in big data services,” in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 393–400.
- [86] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai, “Hibench: A representative and comprehensive hadoop benchmark suite,” in *Proc. ICDE Workshops*, 2010.

- [87] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2007.
- [88] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: Time to rethink?" in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013.
- [89] J. Hsu, "Ibm is redesigning supercomputers to solve big data problems," <https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-redesigned-supercomputers-to-solve-big-data-problems>.
- [90] S. Krishnan, M. Tatineni, and C. Baru, "myhadoop-hadoop-on-demand on traditional hpc resources," *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.
- [91] J. Min, H. Ryu, K. La, and J. Kim, "Abc: dynamic configuration management for microbrick-based cloud computing systems," in *Proceedings of the Posters & Demos Session*. ACM, 2014, pp. 25–26.
- [92] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants," *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2009.
- [93] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts *et al.*, "Gage: A critical evaluation of genome assemblies and assembly algorithms," *Genome research*, vol. 22, no. 3, pp. 557–567, 2012.
- [94] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White, "Low-power amdahl-balanced blades for data intensive computing," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 71–75, 2010.
- [95] A. K. Das, J. Hong, S. Goswami, R. Platania, K. Lee, W. Chang, S.-J. Park, and L. Liu, "Augmenting amdahl's second law: A theoretical model to build cost-effective balanced hpc infrastructure for data-driven science," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 147–154.
- [96] A. Verma, L. Cherkasova, and R. H. Campbell, "Play it again, simmr!" in *IEEE International Conference on Cluster Computing*, 2011.
- [97] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 2010.
- [98] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups." in *MASCOTS*, 2009.

- [99] X. Wu, Y. Liu, and I. Gorton, “Exploring performance models of hadoop applications on cloud architecture,” in *11th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015.
- [100] E. Vianna, G. Comarela, T. Pontes, J. Almeida, V. Almeida, K. Wilkinson, H. Kuno, and U. Dayal, “Analytical performance models for mapreduce workloads,” *International Journal of Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.
- [101] S. Ahn and S. Park, “An analytical approach to evaluation of ssd effects under mapreduce workloads,” *JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE*, vol. 15, no. 5, pp. 511–518, 2015.
- [102] G. Bell, J. Gray, and A. Szalay, “Petascale computations systems: Balanced cyberinfrastructure in a data-centric world,” 2005.
- [103] D. Cohen, F. Petrini, M. D. Day, M. Ben-Yehuda, S. W. Hunter, and U. Cummings, “Applying amdahl’s other law to the data center,” *IBM Journal of Research and Development*, vol. 53, no. 5, pp. 5–1, 2009.
- [104] J. Chang, K. T. Lim, J. Byrne, L. Ramirez, and P. Ranganathan, “Workload diversity and dynamics in big data analytics: implications to system designers,” in *Proceedings of the 2nd Workshop on Architectures and Systems for Big Data*. ACM, 2012, pp. 21–26.
- [105] L. Dobos, I. Csabai, A. S. Szalay, T. Budavári, and N. Li, “Graywulf: A platform for federated scientific databases and services,” in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*. ACM, 2013, p. 30.
- [106] N. J. Gunther, “A simple capacity model of massively parallel transaction systems,” in *CMG-CONFERENCE- COMPSCER MEASUREMENT GROUP INC*, 1993, pp. 1035–1035.
- [107] J. Benet, “Ipfsc-content addressed, versioned, p2p file system,” *arXiv preprint arXiv:1407.3561*, 2014.
- [108] J. Ray, “Whisper overview,” <https://github.com/ethereum/wiki/wiki/Whisper-Overview>.
- [109] T. O. of the National Coordinator for Health Information Technology (ONC). (2015) Report on health information blocking. [Online]. Available: https://www.healthit.gov/sites/default/files/reports/info_blocking_040915.pdf
- [110] A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman, “A case study for blockchain in healthcare: medrec prototype for electronic health records and medical research data,” 2016.

- [111] N. Rifi, E. Rachkidi, N. Agoulmine, and N. C. Taher, "Towards using blockchain technology for iot data access protection," in *Ubiquitous Wireless Broadband (ICUWB), 2017 IEEE 17th International Conference on*. IEEE, 2017, pp. 1–5.
- [112] M. Turkanović, M. Hölbl, K. Košič, M. Heričko, and A. Kamišalić, "Eductx: A blockchain-based higher education credit platform," *IEEE Access*, 2018.
- [113] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6, pp. 1211–1220, 2017.
- [114] D. J. Bodas-Sagi and J. M. Labeaga, "Big data and health economics: Opportunities, challenges and risks," *International Journal of Interactive Multimedia and Artificial Intelligence*, no. In Press.
- [115] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *e-Health Networking, Applications and Services (Healthcom), 2016 IEEE 18th International Conference on*. IEEE, 2016, pp. 1–3.
- [116] A. Gropper. (2016) Powering the physician-patient relationship with the help of one blockchain health it. [Online]. Available: <https://www.healthit.gov/sites/default/files/7-29-poweringthephysician-patientrelationshipwithblockchainhealthit.pdf>
- [117] Z. Alhadhrami, S. Alghfeli, M. Alghfeli, J. A. Abedlla, and K. Shuaib, "Introducing blockchains for healthcare," in *Electrical and Computing Technologies and Applications (ICECTA), 2017 International Conference on*. IEEE, 2017, pp. 1–4.
- [118] I. G. B. S. P. S. Team. (2016) Blockchain: The chain of trust and its potential to transform healthcare our point of view. [Online]. Available: https://www.healthit.gov/sites/default/files/8-31-blockchain-ibm_ideation-challenge_aug8.pdf
- [119] R. Krawiec, D. Housman, F. M. White, Mark, F. Quarre, D. Barr, A. Nesbitt, K. Fedosova, J. Killmeyer, A. Israel, and L. Tsai. (2016) Blockchain a new model for health information exchanges. [Online]. Available: <https://www2.deloitte.com/content/dam/Deloitte/us/Documents/public-sector/us-blockchain-opportunities-for-health-care.pdf>
- [120] C. Brodersen, B. Kalis, C. Leong, E. Mitchell, Eand Pupo, and A. Truscott. (2016) Blockchain: Securing a new health interoperability experience. [Online]. Available: https://www.healthit.gov/sites/default/files/2-49-accenture_onc_blockchain_challenge_response_august8_final.pdf

- [121] T.-T. Kuo, C.-N. Hsu, and L. Ohno-Machado. (2016) Modelchain: Decentralized privacy-preserving healthcare predictive modeling framework on private blockchain networks. [Online]. Available: <https://www.healthit.gov/sites/default/files/10-30-ucsd-dbmi-onc-blockchain-challenge.pdf>
- [122] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, “Med-share: Trust-less medical data sharing among cloud service providers via blockchain,” *IEEE Access*, vol. 5, pp. 14 757–14 767, 2017.
- [123] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, “Integrating blockchain for data sharing and collaboration in mobile healthcare applications,” in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*. IEEE, 2017, pp. 1–5.
- [124] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, “Blockbench: A framework for analyzing private blockchains,” *arXiv preprint arXiv:1703.04057*, 2017.
- [125] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, “Bigchaindb: A scalable blockchain database,” 2016.

Appendix:

Copyright and Permissions Information

5/28/2018

Mail - adas7@lsu.edu

Re: ISCA Website email

isca@isca-hq.org

Sat 5/26/2018 5:51 PM

To: Arghya K Das <adas7@lsu.edu>;

Yes, you may use the conference published paper for your thesis only.

Thanks.

N. Debnath

On 2018-05-25 19:11, adas7@lsu.edu wrote:

From: Arghya Kusum Das
Email: adas7@lsu.edu

Message:

Sub: Requesting Copyright Permission for papers published in BICOB-2017 for PhD Thesis

Dear Sir/Madam,

I would like to followup on our publications titled "GiGA: Giraph-based Genome Assembler for Gigabase Scale Genome" and "ParSECH: Parallel Sequencing Error Correction with Hadoop for Large-Scale Genome Sequences" that we published in BICOB 2016 and BICOB 2017 respectively.

Would you be kind enough to let me know the process for obtaining the copyright permission for using these papers on my final PhD thesis?

Thanks and Regards,
Arghya Kusum Das

<https://outlook.office.com/owa/?realm=lsu.edu&vd=mail>

1/1

**World Scientific Publishing Co., Inc. LICENSE
TERMS AND CONDITIONS**

May 25, 2018

This is a License Agreement between Louisiana State University -- Arghya Das ("You") and World Scientific Publishing Co., Inc. ("World Scientific Publishing Co., Inc.") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by World Scientific Publishing Co., Inc., and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	4353680266526
License date	May 18, 2018
Licensed content publisher	World Scientific Publishing Co., Inc.
Licensed content title	Journal of bioinformatics and computational biology
Licensed content date	Jan 1, 2003
Type of Use	Thesis/Dissertation
Requestor type	Author of requested content
Format	Electronic
Portion	chapter/article
The requesting person/organization is:	Arghya Kusum Das
Title or numeric reference of the portion(s)	PhD Student, First Author of the paper
Title of the article or chapter the portion is from	Large-scale parallel genome assembler over cloud computing environment
Editor of portion(s)	Limsoon Wong
Author of portion(s)	Arghya Kusum Das
Volume of serial or monograph.	Volume:15 Number:03
Page range of the portion	
Publication date of portion	Jun 2017
Rights for	Main product
Duration of use	Life of current edition
Creation of copies for the disabled	yes
With minor editing privileges	yes
For distribution to	Worldwide
In the following language(s)	Original language of publication
With incidental promotional use	no
The lifetime unit quantity of new product	Up to 499

5/25/2018

RightsLink Printable License

Title Large-scale parallel genome assembler over cloud computing environment
Instructor name Seung-Jong Park
Institution name Louisiana State University
Expected presentation date Aug 2018
Billing Type Invoice
Billing Address Louisiana State University
4513 YA Tittle Avenue
Unit 16
Baton Rouge, LA 70820
United States
Attn: Arghya K Das

Total (may include CCC user fee) 0.00 USD

[Terms and Conditions](#)

TERMS AND CONDITIONS

The following terms are individual to this publisher:

None

Other Terms and Conditions:

STANDARD TERMS AND CONDITIONS

1. Description of Service; Defined Terms. This Republication License enables the User to obtain licenses for republication of one or more copyrighted works as described in detail on the relevant Order Confirmation (the "Work(s)"). Copyright Clearance Center, Inc. ("CCC") grants licenses through the Service on behalf of the rightsholder identified on the Order Confirmation (the "Rightsholder"). "Republication", as used herein, generally means the inclusion of a Work, in whole or in part, in a new work or works, also as described on the Order Confirmation. "User", as used herein, means the person or entity making such republication.

2. The terms set forth in the relevant Order Confirmation, and any terms set by the Rightsholder with respect to a particular Work, govern the terms of use of Works in connection with the Service. By using the Service, the person transacting for a republication license on behalf of the User represents and warrants that he/she/it (a) has been duly authorized by the User to accept, and hereby does accept, all such terms and conditions on behalf of User, and (b) shall inform User of all such terms and conditions. In the event such person is a "freelancer" or other third party independent of User and CCC, such party shall be deemed jointly a "User" for purposes of these terms and conditions. In any event, User shall be deemed to have accepted and agreed to all such terms and conditions if User republishes the Work in any fashion.

3. Scope of License; Limitations and Obligations.

3.1 All Works and all rights therein, including copyright rights, remain the sole and exclusive property of the Rightsholder. The license created by the exchange of an Order Confirmation (and/or any invoice) and payment by User of the full amount set forth on that document includes only those rights expressly set forth in the Order Confirmation and in these terms and conditions, and conveys no other rights in the Work(s) to User. All rights not expressly granted are hereby reserved.

3.2 General Payment Terms: You may pay by credit card or through an account with us payable at the end of the month. If you and we agree that you may establish a standing account with CCC, then the following terms apply: Remit Payment to: Copyright Clearance Center, 29118 Network Place, Chicago, IL 60673-1291. Payments Due: Invoices are payable upon their delivery to you (or upon our notice to you that they are available to you for downloading). After 30 days, outstanding amounts will be subject to a service charge of 1-1/2% per month or, if less, the maximum rate allowed by applicable law. Unless otherwise

<https://s100.copyright.com/CustomAdmin/PLF.jsp?ref=e2ea38ac-682b-40ce-9eb1-fc902e8ac89>

2/5

specifically set forth in the Order Confirmation or in a separate written agreement signed by CCC, invoices are due and payable on "net 30" terms. While User may exercise the rights licensed immediately upon issuance of the Order Confirmation, the license is automatically revoked and is null and void, as if it had never been issued, if complete payment for the license is not received on a timely basis either from User directly or through a payment agent, such as a credit card company.

3.3 Unless otherwise provided in the Order Confirmation, any grant of rights to User (i) is "one-time" (including the editions and product family specified in the license), (ii) is non-exclusive and non-transferable and (iii) is subject to any and all limitations and restrictions (such as, but not limited to, limitations on duration of use or circulation) included in the Order Confirmation or invoice and/or in these terms and conditions. Upon completion of the licensed use, User shall either secure a new permission for further use of the Work(s) or immediately cease any new use of the Work(s) and shall render inaccessible (such as by deleting or by removing or severing links or other locators) any further copies of the Work (except for copies printed on paper in accordance with this license and still in User's stock at the end of such period).

3.4 In the event that the material for which a republication license is sought includes third party materials (such as photographs, illustrations, graphs, inserts and similar materials) which are identified in such material as having been used by permission, User is responsible for identifying, and seeking separate licenses (under this Service or otherwise) for, any of such third party materials; without a separate license, such third party materials may not be used.

3.5 Use of proper copyright notice for a Work is required as a condition of any license granted under the Service. Unless otherwise provided in the Order Confirmation, a proper copyright notice will read substantially as follows: "Republished with permission of [Rightsholder's name], from [Work's title, author, volume, edition number and year of copyright]; permission conveyed through Copyright Clearance Center, Inc. " Such notice must be provided in a reasonably legible font size and must be placed either immediately adjacent to the Work as used (for example, as part of a by-line or footnote but not as a separate electronic link) or in the place where substantially all other credits or notices for the new work containing the republished Work are located. Failure to include the required notice results in loss to the Rightsholder and CCC, and the User shall be liable to pay liquidated damages for each such failure equal to twice the use fee specified in the Order Confirmation, in addition to the use fee itself and any other fees and charges specified.

3.6 User may only make alterations to the Work if and as expressly set forth in the Order Confirmation. No Work may be used in any way that is defamatory, violates the rights of third parties (including such third parties' rights of copyright, privacy, publicity, or other tangible or intangible property), or is otherwise illegal, sexually explicit or obscene. In addition, User may not conjoin a Work with any other material that may result in damage to the reputation of the Rightsholder. User agrees to inform CCC if it becomes aware of any infringement of any rights in a Work and to cooperate with any reasonable request of CCC or the Rightsholder in connection therewith.

4. Indemnity. User hereby indemnifies and agrees to defend the Rightsholder and CCC, and their respective employees and directors, against all claims, liability, damages, costs and expenses, including legal fees and expenses, arising out of any use of a Work beyond the scope of the rights granted herein, or any use of a Work which has been altered in any unauthorized way by User, including claims of defamation or infringement of rights of copyright, publicity, privacy or other tangible or intangible property.

5. Limitation of Liability. UNDER NO CIRCUMSTANCES WILL CCC OR THE RIGHTSHOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES (INCLUDING WITHOUT LIMITATION DAMAGES FOR LOSS OF BUSINESS PROFITS OR INFORMATION, OR FOR BUSINESS INTERRUPTION) ARISING OUT OF THE USE OR INABILITY TO USE A WORK, EVEN IF ONE OF THEM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH

DAMAGES. In any event, the total liability of the Rightsholder and CCC (including their respective employees and directors) shall not exceed the total amount actually paid by User for this license. User assumes full liability for the actions and omissions of its principals, employees, agents, affiliates, successors and assigns.

6. Limited Warranties. THE WORK(S) AND RIGHT(S) ARE PROVIDED "AS IS". CCC HAS THE RIGHT TO GRANT TO USER THE RIGHTS GRANTED IN THE ORDER CONFIRMATION DOCUMENT. CCC AND THE RIGHTSHOLDER DISCLAIM ALL OTHER WARRANTIES RELATING TO THE WORK(S) AND RIGHT(S), EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. ADDITIONAL RIGHTS MAY BE REQUIRED TO USE ILLUSTRATIONS, GRAPHS, PHOTOGRAPHS, ABSTRACTS, INSERTS OR OTHER PORTIONS OF THE WORK (AS OPPOSED TO THE ENTIRE WORK) IN A MANNER CONTEMPLATED BY USER; USER UNDERSTANDS AND AGREES THAT NEITHER CCC NOR THE RIGHTSHOLDER MAY HAVE SUCH ADDITIONAL RIGHTS TO GRANT.

7. Effect of Breach. Any failure by User to pay any amount when due, or any use by User of a Work beyond the scope of the license set forth in the Order Confirmation and/or these terms and conditions, shall be a material breach of the license created by the Order Confirmation and these terms and conditions. Any breach not cured within 30 days of written notice thereof shall result in immediate termination of such license without further notice. Any unauthorized (but licensable) use of a Work that is terminated immediately upon notice thereof may be liquidated by payment of the Rightsholder's ordinary license price therefor; any unauthorized (and unlicensable) use that is not terminated immediately for any reason (including, for example, because materials containing the Work cannot reasonably be recalled) will be subject to all remedies available at law or in equity, but in no event to a payment of less than three times the Rightsholder's ordinary license price for the most closely analogous licensable use plus Rightsholder's and/or CCC's costs and expenses incurred in collecting such payment.

8. Miscellaneous.

8.1 User acknowledges that CCC may, from time to time, make changes or additions to the Service or to these terms and conditions, and CCC reserves the right to send notice to the User by electronic mail or otherwise for the purposes of notifying User of such changes or additions; provided that any such changes or additions shall not apply to permissions already secured and paid for.

8.2 Use of User-related information collected through the Service is governed by CCC's privacy policy, available online here:

<http://www.copyright.com/content/cc3/en/tools/footer/privacypolicy.html>.

8.3 The licensing transaction described in the Order Confirmation is personal to User. Therefore, User may not assign or transfer to any other person (whether a natural person or an organization of any kind) the license created by the Order Confirmation and these terms and conditions or any rights granted hereunder; provided, however, that User may assign such license in its entirety on written notice to CCC in the event of a transfer of all or substantially all of User's rights in the new material which includes the Work(s) licensed under this Service.

8.4 No amendment or waiver of any terms is binding unless set forth in writing and signed by the parties. The Rightsholder and CCC hereby object to any terms contained in any writing prepared by the User or its principals, employees, agents or affiliates and purporting to govern or otherwise relate to the licensing transaction described in the Order Confirmation, which terms are in any way inconsistent with any terms set forth in the Order Confirmation and/or in these terms and conditions or CCC's standard operating procedures, whether such writing is prepared prior to, simultaneously with or subsequent to the Order Confirmation, and whether such writing appears on a copy of the Order Confirmation or in a separate instrument.

8.5 The licensing transaction described in the Order Confirmation document shall be governed by and construed under the law of the State of New York, USA, without regard to the principles thereof of conflicts of law. Any case, controversy, suit, action, or proceeding arising out of, in connection with, or related to such licensing transaction shall be brought, at CCC's sole discretion, in any federal or state court located in the County of New York, State of New York, USA, or in any federal or state court whose geographical jurisdiction covers the location of the Rightsholder set forth in the Order Confirmation. The parties expressly submit to the personal jurisdiction and venue of each such federal or state court. If you have any comments or questions about the Service or Copyright Clearance Center, please contact us at 978-750-8400 or send an e-mail to info@copyright.com.

v 1.1

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.



RightsLink®

[Home](#)
[Create Account](#)
[Help](#)


Title: Evaluating different distributed-cyber-infrastructure for data and compute intensive scientific application

Conference Proceedings: 2015 IEEE International Conference on Big Data (Big Data)

Author: Arghya Kusum Das; Seung-Jong Park; Jaeki Hong; Wooseok Chang

Publisher: IEEE

Date: Oct. 29 2015-Nov. 1 2015

Copyright © 2015, IEEE

LOGIN

If you're a [copyright.com user](#), you can login to RightsLink using your copyright.com credentials. Already a [RightsLink user](#) or want to [learn more?](#)

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

[BACK](#)
[CLOSE WINDOW](#)

Copyright © 2018 [Copyright Clearance Center, Inc.](#) All Rights Reserved. [Privacy statement](#). [Terms and Conditions](#).
Comments? We would like to hear from you. E-mail us at customercare@copyright.com



RightsLink®

[Home](#)
[Create Account](#)
[Help](#)


Title: Augmenting Amdahl's Second Law: A Theoretical Model to Build Cost-Effective Balanced HPC Infrastructure for Data-Driven Science

Conference Proceedings: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)

Author: Arghya Kusum Das; Jaeki Hong; Sayan Goswami; Richard Platania; Kisung Lee; Wooseok Chang; Seung-Jong Park; Ling Liu

Publisher: IEEE

Date: 25-30 June 2017

Copyright © 2017, IEEE

LOGIN

If you're a [copyright.com user](#), you can login to RightsLink using your [copyright.com](#) credentials.

Already a [RightsLink user](#) or want to [learn more?](#)

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

[BACK](#)
[CLOSE WINDOW](#)

Copyright © 2018 [Copyright Clearance Center, Inc.](#) All Rights Reserved. [Privacy statement](#). [Terms and Conditions](#).
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Vita

Arghya Kusum Das was born in West Bengal, India, in 1986. He obtained his bachelors degree (Bachelor of Technology) in Computer Science and Engineering in 2008 from West Bengal University of Technology. Prior to joining the Doctoral study at Louisiana State University, Das has multiple years of industry experience in Wipro Technologies (Hyderabad, India) and Matrix Educare Pvt. Ltd. (Kolkata, India). He also worked as Junior Research Fellow at Indian Institute of Engineering Science and Technology (Shibpur, India).

During the last few years of Doctoral study at Louisiana State University, Das has worked as research assistant at the Center for Computation and Technology of the University. During his doctoral studies at Louisiana State University, Das has worked on multidisciplinary research projects involving large-scale scientific datasets in the domain of bioinformatics, biomedical and environmental science. Many of the observations, analysis and developments of his Doctoral thesis have been used in procuring and setting up IBM Power8-based high performance computing cluster called Delta at Louisiana State University.