Feature Enhancement and Performance Evaluation of BioPig Analytics

by

Lizhen Shi

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 12, 2015

Keywords: Bioinformatics, Hadoop, Pig, k-mer, Tuning

Approved by

Weikuan Yu, Chair, Associate Professor of Computer Science and Software Engineering
Sanjeev Baskiyar, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering

Abstract

Next-Generation sequencing produces huge collections of strings to be analyzed. This massive dataset challenges traditional analytics tools and increasingly requires novel solutions adapting to big data platforms. MapReduce software framework presents a viable solution to large-scale sequence analysis in terms of efficiency and scalability. Hadoop as an open-source implementation of MapReduce framework is designed to run applications on large-scale clusters built on commodity hardware. Hadoop distributed file system (HDFS) and Hadoop MapReduce are two important components of Hadoop framework. HDFS provides scalable, fault-tolerant and distributed data storage, while MapReduce is the core concept of Hadoop framework and provides a scale-out data processing solution across hundreds or thousands of nodes in Hadoop cluster. Since Hadoop version 0.23, MapReduce has changed significantly, which we call MapReduce 2 (aka YARN). YARN is the resource manager of the Hadoop cluster and has the ability to enhance the power of cluster computation. Hadoop is being widely used in many domains including Bioinformatics.

BioPig, the current version of which is built on Hadoop 1, is a Hadoop-based toolkit for large-scale sequence analysis. In this thesis, I present the YARN-based BioPig toolkit, which is an upgrade and continuous development of current version of BioPig. Benefits are gained from the development: not only the job throughput and cluster utilization are improved, but also other computational frameworks are permitted to run on Hadoop cluster simultaneously.

k-mer counting is a preliminary step of subsequent sequence analyses in Bioinformatics and acts as a central role in BioPig. Unlike usual application workloads, k-mer counting generates a large volume of intermediate data which makes general parameter tuning guidelines inapplicable. To optimize BioPig performance on YARN cluster, I tuned Hadoop parameters according to the distinct k-mer counting workload characteristic from five perspectives: data

compression, HDFS block size, map-side spills, JVM garbage collection and reducer start-time. The evaluation reveals that this tuning practice reaches a significant performance gain comparing to the performance of the baseline configuration: the overall job execution time is reduced by about 50% . Through feature enhancement and performance evaluation, this thesis provides a valuable reference for other similar applications that generate large volume of intermediate data. Besides migrating current BioPig to YARN and tuning parameters, I also developed a new module, PigSimilarity, to extend the application domain of BioPig tookit.

Acknowledgments

Along the journey of my studies at the Computer Science and Software Engineering (CSSE) master program at Auburn University, I have been encouraged, supported and inspired by many people. Without their help, it would be impossible for me to complete the program within one year. Here, I would like to take the opportunity to express my thanks to several people for their contribution to the development and completion of the thesis.

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Weikuan Yu, for the financial support of my master study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Baskiyar and Prof. Biaz for their encouragement, insightful comments and questions.

My sincere thanks also goes to Dr. Zhong Wang for offering me the summer internship opportunity in their team and leading me working on diverse exciting projects in bioinformatics.

I thank my fellow labmates in Parallel Architecture & System Laboratory (PASL) : Dr. Hui Chen, Dr. Jianhui Yue, Bin Wang, Cong Xu, Zhuo Liu,Teng Wang, Huansong Fu, Fang Zhou Xinning Wang, Kevin Vasko, Michael Pritchard, Hai Pham, Bhavitha Ramaiahgari, Yue Zhu, and Hao Zou, for the technical communication and discussion, for the sleepless night we were working together before the deadline, for the fun we were talking and playing together. PASL is a big family for me. I am so happy that I can be a member of it. The period of time spent in PASL is my forever wealth in my life.

Furthermore, my warmest thanks must be to my family. Thanks to my parents-in-law for taking care of my baby during my study time at Auburn University. Thanks to Chloe Chen, the best daughter I could ever have, for her smiles encourage me to efficiently overcome the difficulties encountered in my pursuit of the MA degree. Lastly, thanks to my dear husband, for his continued and unfailing love, support and understanding underpins my persistence in the graduate career and makes the completion of this thesis possible.

Last but not least, I would like to acknowledge the sponsors of this research and my graduate study at Auburn University. Particularly, I would like to acknowledge the National Science Foundation awards 1059376, 1320016, 1340947 and 1432892.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In just several years, next-generation technologies have reduced the cost and increased the speed of DNA sequencing by four orders of magnitude. Back many years ago, reading DNA was a 3 billion long step job. Automation and computerization revolutionized the speed of reading the letters of DNA sequences first. And then the advent of next-generation sequencing technique increased it further. Up to now, modern Illumina systems can generate 6 hundreds of gigabytes per run [34] with 99.9% accuracy. As a result, this rapid pace of sequencer improvement gives rise to large amount of raw sequence data to be processed and hence places huge burden on external compute framework.

Meanwhile, extremely large scale sequencing projects are emerging, such as ENCODE project, 1000 Genomes project, Cow Rumen Deep Metagenomes project and Human Microbiome project. These projects produce Next-Gen sequencing data on a massive scale. Traditional analysis tools have got challenged by the fact that they have difficulty in scaling with data size.

Facing Next-Generation sequencing platform improvement and these emerging large scale sequencing projects, the life science has been transformed into a data-intensive field. Parallel processing paradigm (e.g. MPI) and graphical processing unit (GPU) are used to efficiently process large-scale sequencing datasets. However, both of these two models have limitations, which are described in great details in the later background part.

Hadoop [2] platform emerged as an economically viable option to address the challenges of increasingly large datasets. Hadoop distributed file system (HDFS) and MapReduce are two core components of Hadoop framework. With the assistance of HDFS, Apache Hadoop not only enables distributed and fault tolerance large dataset storage, but also built-in data

locality and distributed data processing. MapReduce programming model permits tasks running in a massively parallel way on large number of nodes. With the combination of MapReduce and HDFS, Hadoop is considered as a load balancing, scalable and reliable framework. Nowadays it becomes widely adopted and used in many scientific domains.

BioPig [25] is one of the Hadoop-based toolkits for large-scale NGS analysis in Bioinformatics. It is fully open source under the BSD license, and is implemented on top of Apaches Hadoop framework and Pig data flow language. Pig Latin [26], an expressive and flexible language for data analysis, eases the development of sequence analysis tools in Bioinformatics using Hadoop framework. Leveraging the advantages of Hadoop and Pig framework, BioPig toolkit has both the scalability and robustness offered by Apache Hadoop and the programmability offered by Pig. In addition, since both Hadoop and Pig are implemented in Java, BioPig inherits their portability.

However, current BioPig has limitations because it was built on earlier Hadoop version (v0.20). Hadoop framework has undergone a big architecture shift since version 0.23, the new version of Hadoop is called Hadoop 2. HDFS federation and YARN [8] resource manager are two significant advances in Hadoop 2, which were designed to overcome issues of lacking scalability and high-availability in Hadoop 1.

In this paper, I first introduced the YARN-based BioPig which is migrated from current BioPig. The new BioPig, inheriting all the Hadoop 2 features, not only scales efficiently on modern cloud computing platform but also permits other computational frameworks (e.g. MPI, Spark [4] etc) to run on the same cluster.

Secondly based on the workload characteristics of BioPig functions, I tuned Hadoop parameters from 5 perspectives: data compression, HDFS block size, map-side spills, JVM garbage collection and reducer start-time, to optimize the YARN-based BioPig performance. The overall job execution time decreased by about 50% after these delicate tuning steps were applied.

Last I implemented another widely used function in Bioinformatics, which is calculating pairwise sequence distance among a set of sequences, to extend BioPig framework. Multiple existing tools can do the same job. However BioPig functions scale well with the data size and k-mer size by leveraging the Hadoop framework.

The thesis is organized as follows:

- Chapter 2 introduces background of Bioinformatics and Hadoop for the BioPig tookit.
- Chapter 3 describes main modules in current BioPig framework.
- Chapter 4 talks about the implementation including both Hadoop/Pig upgrade and new module development.
- Chapter 5 discusses the testing environment setup including EC2 cluster configuration, input data preparation and baseline settings of Hadoop. The baseline performance is also presented.
- Chapter 6, based on the baseline settings, tunes Hadoop parameters from five perspectives. The final tuned performance is presented and compared with baseline performance.
- Chapter 7, the last part, discusses the conclusion and future work.

Chapter 2

Background

In this chapter, I first compare Hadoop framework with two other parallel computing models: MPI (Message Passing Interface) and GPU (Graphics Processing Unit). The purpose is to demonstrate Hadoop's advantages over other parallel computing models. Then I introduce the main components in Hadoop - Hadoop Distributed File System (HDFS) and MapReduce framework - as well as the major differences between Hadoop 1 and Hadoop 2. In the third section, I talks about the widely adoption of MapReduce programming model in Bioinformatics and some available MapReduce-based tools. Then k-mer counting and its implementation in MapReduce Paradigm are described next. Finally Cloud computing and Elastic Computing Cloud (EC2) are introduced.

2.1 Parallel Computing Models Comparison

MPI, GPU and Hadoop are three mostly used parallel computing models to process large volume of data in various scientific domains. Running Hadoop program with a limited amount of data on a small number of nodes does not yield any benefit but the overhead of starting Java processes. In this case MPI and GPU demonstrate much better performance. However as data size increases significantly, the disadvantages of MPI and GPU emerges.

Specifically MPI program is considered to have the following disadvantages:

1. MPI program runs parallel on worker nodes. When jobs finish, master node combines the results from the worker nodes to deliver the final result. All the worker processes run copies of the same program, and they communicate with each other through messages. When the number of nodes scales up to a certain threshold, the network bandwidth

becomes the bottleneck. In other words MPI distributes program but data. The lack of data distribution prevents it to scale as data size increases.

2. As cluster gets larger, fault tolerance becomes a more important issue. Current MPI implementations do not provide any mechanism of fault tolerance. Program simply crashes when hardware fails.

3. MPI program may require specific refactoring for large cluster.

4. Researchers with a Biology background consider MPI program too complicate to develop.

The Compute Unified Device Architecture (CUDA) parallel computing platform provides significantly performance boosting by using the state of art GPU architecture. A GPU consisting of hundreds of cores allows certain operations processing data concurrently. Hence this programming model is suitable for computation-intensive Bioinformatics tasks for which an operation need to be applied to many similar records. However, the limitation of shared memory and global memory in a GPU is a fatal flaw when scaling with the number of records.

Unlike MPI and GPU, Hadoop provides linear scalability with data size. When data size is significantly large, the overhead of spawning processes, querying data chunk location, tracking the work processes, is completely insignificant comparing to the task execution cost. Hadoop also eases development - a program runs in a single process can run successfully on any size of cluster without any modification. Hardware and network failure are also handled automatically by Hadoop framework.

2.2 Overview of Hadoop Framework

Apache Hadoop as an open source implementation of MapReduce paradigm provides a reliable, scalable and distributed computing framework for data intensive applications. The project includes 3 modules: Hadoop Common, Hadoop distributed File system (HDFS) and

Hadoop MapReduce. Hadoop Common provides a set of utilities and libraries that support other Hadoop modules. HDFS is a distributed file system that links together the file systems on the Hadoop data nodes. It provides reliable data storage and high-throughput data access for all Hadoop applications. MapReduce is the heart concept of Hadoop framework and provides a scale-out data processing solution across hundreds or thousands of servers in a Hadoop cluster.

2.2.1 Hadoop Distributed File System (HDFS)

Using low cost commodity hardware in a very large scale is one key difference of Hadoop Framework from other parallel computing paradigms. Data to be processed in the cluster is broken into blocks and is distributed stored in the inexpensive disk drives of the data nodes, where hardware failure occurs pretty common. To achieve high availability, each block of data is replicated to 3 machines by default to prevent the failure of one machine from losing all copies of data. To improve data processing performance, HDFS supports data locality, which means the workload would be assigned to the node where the data to be processed is stored. This technology decreases network overhead and provides Hadoop cluster the ability of linear scalability.

HDFS includes components of NameNode and DataNode. Even though NameNode does not hold any cluster data, it holds the file system metadata for the Hadoop cluster and monitors the health of the DataNodes through the heartbeats from DataNodes. As a result NameNode is the center controller and the most critical component of HDFS. Prior to Hadoop 2.0, HDFS architecture allows only a single namespace for the entire cluster and the cluster size is limited because of the single NameNode. Since Hadoop 2.0, Federation is designed to address this limitation, in which architecture allows multiple NameNodes and NameSpaces. HDFS federation enables Hadoop cluster to scale horizontally. The HDFS architecture difference between Hadoop 1 and Hadoop 2 is shown in Figure 2.1.

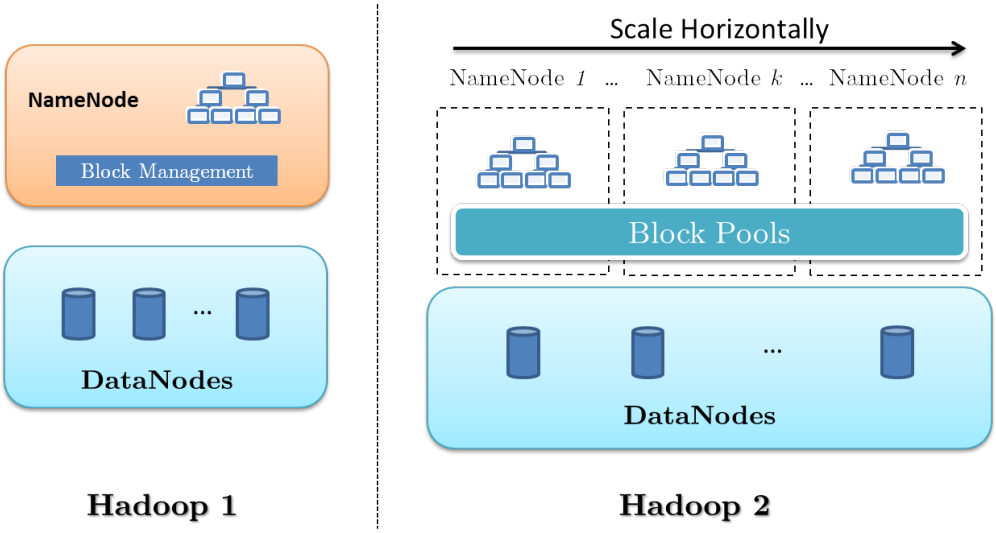

Figure 2.1: HDFS Architecture Comparison

2.2.2 Hadoop MapReduce

MapReduce [10] reflects a new programming model for large volume of data computing in a parallel fashion and has gained a lot of attention from various scientific communities since it was introduced by Google in 2004. The term MapReduce actually comes from two separate functions that users need to implement: one is map function which takes a set of data and generates intermediate key-value pairs; the other one is reduce function which takes the output from map functions and merges all intermediate values associated with the same key. The reduce function can only take place when the map phase completes, just as the term sequence in "MapReduce" implies. In this model, the workload is decomposed into a large number of small tasks and distributed to large number of nodes. Many (not all) real-world applications tasks fit very well into the MapReduce paradigm and can be executed in this environment. Users may use a general-purpose programming language such as Java or Python to implement their own processing logic by developing customized map and reduce functions.

MapReduce is in its second generation, which we call MapReduce 2 (MRv2) or YARN. Its architecture has undergone a big shift compared to its earlier version. The motivation of the development of YARN was to overcome several limitations in MRv1:

- One of the most important issues in MapReduce 1 is that map and reduce slots are separately pre-configured. This inflexibility leads to an underutilization of cluster hardware resource. In YARN, resources (e.g. memory, cpu, disk, network bandwidth etc) are put into one unit called resource container. Both the map task and reduce task run in containers - there is no fixed usage for a container. In other words, the same container can be used for Map task, Reduce task or other non-mapreduce task. As a result YARN gives much more flexibility in scheduling, which leads to better hardware utilization and brings performance improvement.

- Another limitation of MRv1 is the overburdened JobTracker(Resource management and job scheduling/monitoring). YARN divides the two major responsibilities of the Jobtracker into two separate components: global resource manager(RM) and per-application ApplicationMaster(AM). RM only manages the allocation of resources to the submitted jobs. AM takes care of application life cycle management. This function separation enables Hadoop to run on much larger clusters.

- MRv1 only supports MapReduce type jobs, which makes large amount of data stored in HDFS can only be used for MapReduce processing. YARN is more general and opens the door for Non-MapReduce Big Data applications. Both the MapReduce and Non-MapReduce applications can coexist running on the Hadoop cluster and share the data stored in the HDFS in MRv2

In conclusion, compared to MRv1's restricted batch processing model, MRv2 is more capable and specialized. YARN as the operating system of Hadoop cluster not only allocates resource more flexible and dynamic, but also extends the Hadoop ecosystem beyond MapReduce and batch processing.

2.3 MapReduce-based Bioinformatics Tools

DNA sequences, which consists of only four unique bases labeled A, T, C, and G (for adenine, thymine, cytosine, and guanine respectively), make up any organism and determine the genetic characters of organisms. DNA sequence analysis is the process of subjecting a DNA to any of a wide range of analytical methods to understand its features, functions, structure, or evolution. Methodologies used include sequence alignment, searches against biological databases, etc [9]. Next Generation Sequencing (NGS) techniques decrease the cost of sequencing and produce huge volumes of DNA sequences every day. Advances in sequencing technologies shift the burden from chemistry done in a laboratory to computational analysis. MapReduce provides an easy-to-use and fault-tolerance parallel programming model for processing petabytes of sequence data on commodity Linux clusters. Hadoop and Spark, as two open-source implementations of the MapReduce framework, have a diverse and growing community in Bioinformatics. Table 2.1 concludes related MapReduce-based Bioinformatics tools in the market.

Name	Year	Description
CloudBLAST [21]	2008	Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications
CloudBurst [28]	2009	Highly sensitive read mapping with MapReduce
Crossbow [17]	2009	Searching for SNPs with cloud computing
GATK [22]	2010	The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data
Myrna [16]	2010	Cloud-scale RNA-sequencing differentialexpression analysis
Galaxy [12]	2010	Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences
SEAL [27]	2011	SEAL: a Distributed Short Read Mapping and Duplicate Removal Tool
CloudAligner [23]	2011	A fast and full-featured MapReduce based tool for sequence mapping
FX [13]	2012	an RNA-Seq analysis tool on the cloud
BioPig [25]	2013	a Hadoop-based analytic toolkit for large-scale sequence data
SeqPig [29]	2014	simple and scalable scripting for large sequencing data sets in Hadoop
SparkSeq [33]	2014	fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision
ADAM [20]	2014	ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing
Halvade [11]	2015	Halvade: scalable sequence analysis with MapReduce

Table 2.1: Available MapReduce-based Bioinformatics Tools

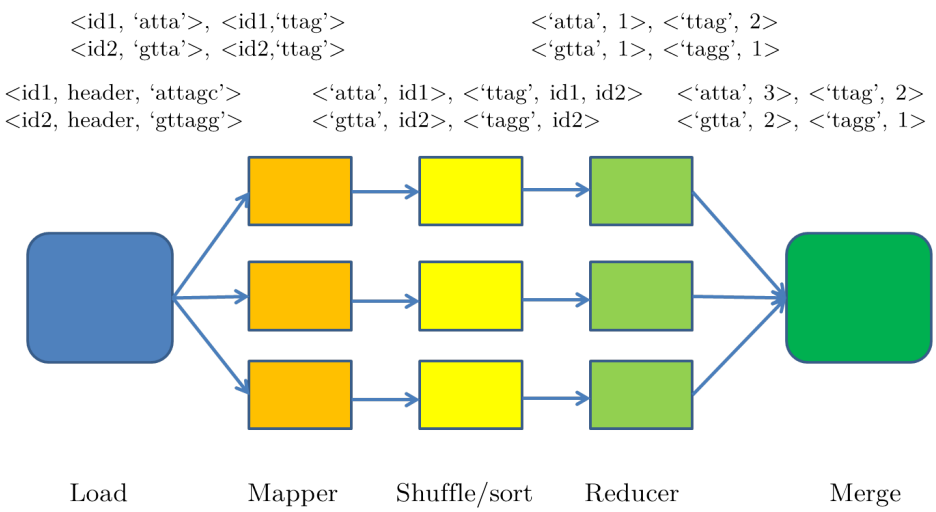

Figure 2.2: k-mer Counting Implementation in MapReduce

2.4 k-mer Counting and its Implementation in MapReduce Paradigm

In Bioinformatics, k-mers refer to all the possible subsequences of length k in a DNA/RNA sequencing read. The k-mer counting problem, which is to count the occurrences of every k-mer in a set of reads from genome sequencing projects, is the preliminary and central step of many subsequent read analyses , including constructing de Bruijn graphs in sequence assembly, elimination of erroneous reads in a relatively large number of dataset and fast multiple sequence alignment. k-mer counting is simple in principle and hence the easiest problem in Bioinformatics. Even novice with a few minutes of training in Perl or Python can implement a k-mer counting function. But when the k-mer size is large and billions of reads need to be processed, k-mer counting becomes the most difficult problem in Bioinformatics. For huge amount of sequence data, as the k-mer size increases the space required to store k-mer counting increase exponentially with k-mer size. Counting large k-mers in large modern sequence data sets can easily overwhelm the memory capacity of standard computers. To address the scalability issue, BioPig framework provides a scalable k-mer counter which scales well with the dataset size and k-mer size due to the linear scalability of Hadoop framework. How k-mer counting is implemented using MapReduce paradigm can be seen in Figure 2.2.

2.5 Cloud Computing in EC2 for Bioinformatics

Cloud computing offers a solution for ever increasing large scale scientific analysis, researchers can lease hardware resource from cloud computing vendors instead of purchasing and maintaining expensive hardware resources. Amazon is the biggest cloud vendor on the market. The Amazon Web Services (AWS) [5] cloud provides a highly reliable and scalable infrastructure for deploying web-scale solutions[32]. Amazon Elastic Compute Cloud (EC2) [7] is a central part of AWS and provides scalable, pay-as-you-go compute capacity. EC2 eases the availability of Hadoop clusters for biological researchers. With customized Amazon machine images (AMIs), the computation environment of a particular application can be saved and then be replicated later. Hence the past result can be easily reproduced. We use EC2 to evaluate our YARN-based BioPig performance. BioPig AMI is also provided to users to simplify the setup process and enhance user experience.

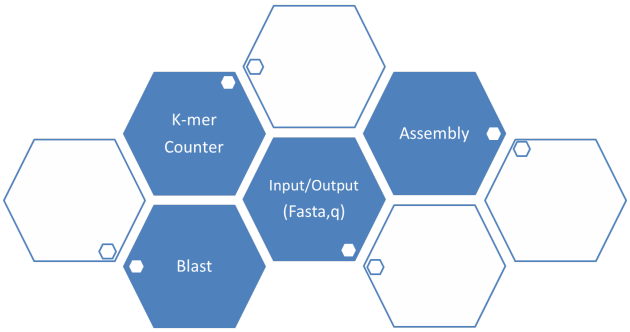

Figure 3.1: BioPig Modules

Chapter 3

BioPig Modules

BioPig is an analytic toolkit for Next-Generation Sequencing analysis and is built on Apache Hadoop framework and Pig Latin data flow language. Hadoop framework utilizes the data-parallel paradigm of MapReduce which was first introduced by Google to distribute computations over many computing nodes. Data locality technique of HDFS takes the computation to the data node and hence avoids network transfer bottleneck for large cluster. With *Pig Latin*, an easy-to-write-and-understand high level language, Bioinformatics researchers can accomplish their jobs by writing their own scripts. Testing shows BioPig framework demonstrates scalability, programmability and portability.

The first release of BioPig consists of four core functional modules: Input/Output, k-mer counter, Blast and Assembly. The Input/Output module takes the fasta and fastaq as input and then splits and distributes the data across the Hadoop cluster. The k-mer counter computes the frequencies of each k-mer in sequence reads and outputs a histogram of the kmer counts. Blast provides a wrapper-based MapReduce implementation of BLAST for Hadoop, Assembly is a wrapper for common Bioinformatics assembler, such as CAP3, Velvet and KmerMatch. The main modules of BioPig are shown in Figure 3.1.

Chapter 4

Implementation

4.1 Development Environment

Development and unit tests can be done on personal desktop or laptop without powerful cluster environment. Linux or Mac operation system is preferred to Windows because the installation of Hadoop/Pig on these systems is easier than on Windows system. The development environment I used is as follows:

- Mac mini, OS X 10.9, 4G
- OpenJDK 7
- maven 3
- Hadoop 2.7.0
- Pig 0.15.0
- Git 1.9.5
- Eclipse 4.5

4.2 Biopig Upgrade

The task includes upgrading Hadoop 1 to Hadoop 2 and upgrading Pig 0.10 to version 0.15. Although Hadoop 2 has evolved a lot compared to Hadoop 1, Hadoop contributors claim binary compatibility of *mapred* API and source compatibility of *mapreduce* API to existing applications. However Pig has to be updated because Pig 0.10 can not run directly on Hadoop 2 due to incompatility of APIs and configuration.

4.2.1 Hadoop Upgrade

Hadoop upgrade is relatively easy since Hadoop 2 are backward compatible with earlier release at the source level. The maven dependency has to be modified to compile against the new version as shown in Code 1 and Code 2. Notice that besides the version numbers, the *artifactedId*'s also changes.

Code 1 Hadoop 1 maven dependency

```
-        <dependency>
-            <groupId>org.apache.hadoop</groupId>
-            <artifactId>hadoop-core</artifactId>
-            <version>0.20.2</version>
-        </dependency>
-        <dependency>
-            <groupId>org.apache.hadoop</groupId>
-            <artifactId>hadoop-tools</artifactId>
-            <version>0.20.2</version>
-        </dependency>
-        <dependency>
-            <groupId>org.apache.hadoop</groupId>
-            <artifactId>hadoop-test</artifactId>
-            <version>0.20.2</version>
-        </dependency>
```

Code 2 Hadoop 2 maven dependency

```
+ <dependency>
+    <groupId>org.apache.hadoop</groupId>
+    <artifactId>hadoop-client</artifactId>
+    <version>2.7.0</version>
+ </dependency>
+ <dependency>
+    <groupId>org.apache.hadoop</groupId>
+    <artifactId>hadoop-minicluster</artifactId>
+    <version>2.7.0</version>
+    <scope>test</scope>
+ </dependency>
```

4.2.2 Pig Upgrade

Pig upgrade includes dependency changes and API changes. Code 3 shows the maven dependency changes.

Code 3 Pig maven dependency changes

```
<!--  Pig 0.10 dependency -->
  <dependency>
-    <groupId>org.apache</groupId>
      <artifactId>pig</artifactId>
-    <version>0.10.0</version>
  </dependency>

<!--  Pig 0.15 dependency -->
  <dependency>
+     <groupId>org.apache.pig</groupId>
      <artifactId>pig</artifactId>
+     <version>0.15.0</version>
  </dependency>
```

There are some API differences between pig 0.10 and pig 0.15. Code 4 shows some of pig-specific classes that were removed in pig 0.15. Because these classes are only used in unit test cases, simply removing them fixes the issue. Code 5 shows another modification of *GruntParser* constructor between these two versions.

Code 4 Pig API's that has been removed

```
-import org.apache.pig.impl.logicalLayer.LogicalPlan;
-import org.apache.pig.impl.logicalLayer.parser.ParseException;
-import org.apache.pig.impl.logicalLayer.parser.QueryParser;
-import org.apache.pig.backend.hadoop.executionengine.
                      physicalLayer.LogToPhyTranslationVisitor;
```

4.3 k-mer Based Distance

Besides Hadoop and Pig upgrades, I also implemented an importance module: pigSimilarity. Given a set of sequences, the pigSimilarity module computes the pairwise sequence

Code 5 GruntParser Interface Changes

```
/* Pig 0.10 code */
-         GruntParser parser = new GruntParser(new StringReader(script));
          parser.setInteractive(false);
-         parser.setParams(ps);

/* Pig 0.15 code */
+       GruntParser parser = new GruntParser(new StringReader(script),ps);
         parser.setInteractive(false);
```

distances. Similarity-based searches and joins are important for many applications such as document clustering or near-duplicate and plagiarism detection [8, 9, 18]. In recent years, much effort has been spent on developing tools to speed up similarity-based queries on sequences. BioPig function can scale well with data size and k-mer size.

4.3.1 Definition

The similarity between two nucleotide sequences can be quantified by the concept of *k-mer based distance*. It is analogous to the distance in physical space. The larger the distance between two sequences, the less their similarity. To be formal, some related definitions are provided bellow.

Definition 4.1. k-mer generating. Given a k-mer size k, k-mer generated set $G_k(s)$ of a nucleotide string s of length n is the set of all possible substrings of s of length k, that is

$$G_k(s) \coloneqq \{t : t \textit{ is a substring of length } k \textit{ of string } s\}$$

Example 4.1. Let $s = CTTCGAAA$, $G_4(s) = \{CTTC,\ TTCG,\ TCGA,\ CGAA,\ GAAA\}$

Definition 4.2. Reverse complement. Let t be a nucleotide string, reverse complement of t, $RC(t)$, is formed by reversing the string and taking the complement of each symbol.

Example 4.2. Let $s = CTTCGAAA$, $RC(s) = TTTCGAAG$

Definition 4.3. Less complement. Let t be a nucleotide string, less complement of t, $LC(t)$, is the smaller one of t and $RC(t)$ in lexicographical order.

Example 4.3. Let $s = CTTCGAAA$, then $RC(s) = TTTCGAAG$. Because $CTTCGAAA$ is less than $TTTCGAAG$ in lexicographical order, $LC(s) = CTTCGAAA$. Also $LC(RC(s)) = CTTCGAAA$

Definition 4.4. k-mer counting. Given a k-mer size k, k-mer counting of string s, $C_k(s)$, is a map of k-mer to integer which represents the occurrence numbers of less complement k-mers in $G_k(s)$.

Example 4.4. Let $s = CTTCGAAA$,

$$C_4(s) = \{CTTC{:}1,\ TCGA{:}1,\ CGAA{:}2,\ GAAA{:}1\}$$

Definition 4.5. k-mer space. Given k-mer size k, let B_k =$\{$ *all the less complement k-mer's of length k* $\}$, n is the size of the B_k and B_k^i is the i-th k-mer in B_k. Mapping B_k onto the basises of an n-dimension space Ω, we say Ω is a k-mer space of k-mer size k and B_k^i is the k-mer label of i-th dimension.

Definition 4.6. Euclidean distance. Let B_k and n be defined as above and $\Omega = \mathbb{R}^n$. $M = C_k(s)$ is the k-mer counting of string s. M can be mapped to a point X in $\mathbb{R}^n$, where $X_i = M[B_k^i]$ for $i = 1, ..., n$. Then length of the k-mer counts can be defined as $l_k^E(s) := \sqrt{\sum_{i=1}^{n} X_i^2}$.
Let Y be the mapping point of string t. The Euclidean distance between string s and t for k-mer size k is defined as:

$$D_k^E(s,t) = \sqrt{\sum_{i=1}^{n} \Big(\frac{X_i}{l_k^E(s)} - \frac{Y_i}{l_k^E(t)}\Big)^2}$$

Definition 4.7. Hamming distance. Let B_k and n be defined as above, $\Omega = \{0,1\}^n$ and $M = C_k(s)$ is the k-mer counts of string s. M can be mapped to a point X in Ω, where

$X_i = \mathbb{1}_{M[B_k^i]>0}$ for $i = 1, ..., n$, where $\mathbb{1}$ is indicator function.

Let Y be the mapping point of string t. The Hamming distance between string s and t for k-mer size k is defined as:

$$D_k^H(s,t) = \sum_{i=1}^{n} |X_i - Y_i|$$

4.3.2 Pig Functions

The advantage of Pig Latin data flow language is that we can implement small functions and then write Pig script to express the data flows using built-in operations, built-in functions and user-defined functions. Two distance functions, Hamming distance and Euclidean distance, were implemented as Pig evaluation functions in BioPig. In this section, we mainly describe user-defined functions.

k-mer generating function

The *k-mer generating function* was implemented in Biopig 1. It is included here for completeness. The function takes DNA sequence and k-mer size as arguments and returns a bag of k-mer strings. The meanings of data types in Pig can be found in Table 4.1

```
gov.jgi.meta.pig.eval.kmerGenerator(seq:chararray, kmer_size:int) as k-mer:bag
```

Data Type	Description
int	Signed 32-bit integer
long	Signed 64-bit integer
float	32-bit floating point
double	64-bit floating point
chararray	Character array (string) in Unicode UTF-8 format
bytearray	Byte array (blob)
tuple	An ordered set of fields.
bag	An collection of tuples.
map	A set of key value pairs.

Table 4.1: Pig Data Types

Hamming distance function

The function takes a 2-tuple of k-mer bags which are the k-mer counts for two sequences and returns the Hamming distance between the two sequences.

```
gov.jgi.meta.pig.eval.TNFHammingDistance(data:tuple) as distance:int
```

Euclidean distance function

The function takes a 2-tuple of k-mer bags which are the k-mer counts for two sequences and returns the Euclidean distance between the two sequences.

```
gov.jgi.meta.pig.eval.TNFDistance(data:tuple) as distance:float
```

4.3.3 Pig Scripts

Given a set of sequences, the pigSimilarity module computes pairwise hamming distance and outputs a matrix of these distance.s The following code list shows the BioPig script for this calculation.

```
1  -- a more elaborate pig script example to calculate pairwise sequence distance of a set of
       sequences
2  -- Command line parameters
3  --   READS = the location of the input file to read.
4  --   OUTPUTDIR = the directory to put the results
5  --   P= the level of parallelism for the reduce, defaults to 10
6  register ../biopig/target/biopig.jar
7  %default READS '../biopig/src/test/resources/1M.fas'
8  %default P '10'
9  %default OUTPUTDIR 'x
10 A = load '$READS' using gov.jgi.meta.pig.storage.FastaStorage as (readid: chararray, d:
       chararray, seq: bytearray);
11 B = foreach A generate readid,gov.jgi.meta.pig.eval.k-merGenerator(seq, 4) as k-mers ;
12 BB = foreach B generate readid, flatten(k-mers);
13 C = foreach BB  generate readid, gov.jgi.meta.pig.eval.UnpackSequence(k-mer) as k-mer;
14 D = foreach ( group C by (readid, k-mer) ) generate flatten(group), COUNT($1) as cnt;
15 E = foreach (GROUP D by $0) {
16 data = foreach $1 generate $1 as k-mer, $2 as cnt;
```

```
17   generate $0 as readid, data;
18  };
19  E2 = foreach E generate *;
20  J = foreach (CROSS E, E2) generate $0 as read1,$2 as read2, ($1,$3) as data;
21  L = FOREACH J GENERATE $0 as read1,$1 as read2, flatten(gov.jgi.meta.pig.eval.
        TNFHammingDistance(data));
22  store L into '$OUTPUTDIR';
```

A few key lines are explained here:

line 10 loads a fasta file located in HDFS or local disk and converts the reads to tuples.

line 11 generates k-mers for each read.

line 14 groups identical k-mers and generates a count of each group.

line 20 makes a cross product of reads to generate pairwise read data.

line 21 calculates the pairwise Hamming distance for all the reads.

Chapter 5

Evaluation

5.1 Hardware and Software Configuration

To evaluate YARN-based BioPig performance, 15 nodes (1 name node and 14 data nodes) of the type c3.8x large instances were launched in EC2. Each node has 60GB RAM and 108 Elastic Computer Units (Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz). A 500GB solid-state drive (SSD) was attached to each node as local and HDFS storage for MapReduce jobs. All the nodes are interconnected by 10Gbit/s Ethernet network. The software configruation is as follows:

- Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD)
- OpenJDK 7
- Hadoop 2.7.0
- Pig 0.15.0

5.2 Input Data

Cow rumen metagenomics dataset, the same dataset used in the original BioPig Paper [25], is chosen as test input. From the I250 serial, the one with the largest size and the best quality fasta file is truncated to generated 7 different workloads: 1GB, 5GB, 10GB, 20GB, 40GB, 60GB and 100GB. The k-mer size is fixed to be 20.

5.3 Baseline Test

5.3.1 Baseline Configuration

Hadoop has a large set of parameters that may be specified to control the behavior of jobs. Of those, about 30 parameters can affect the performance significantly. Default values are used if the parameters are not specified. Using default values most of the time leads to underperformance due to the following reasons:

- First Hadoop 2 was first release in 2013. Hardware resource has been improved a lot during the past two years.
- Secondly every cluster has its own hardware configuration. It is unlikely the default values are applicable to any cluster.
- Thirdly the default values are usually set to be relatively low to avoid exceptions.

We are not going to use these default values as the baseline configuration. Thus the first step is to set these configuration parameters to appropriate values. The goal is to get an acceptable performance providing the limitations of hardware resources (e.g. CPU, memory, disk, network). Having reviewed the best practices of other Hadoop applications, values of some of the parameters are set as shown in table 5.3.1. These parameters phsically exists in hdfs-site.xml, mapred-site.xml and yarn-site.xml accordingly. The differences between baseline settings and default values are also shown in this table.

A few of key points of the configuration are emphasized below:

- Set 2Gb memory limitation to Map containers and Reduce containers.
- Allocate a max Java heap size of 1624Mb for Map/Reduce processes.
- Allocate 1000Mb buffer size for storing map output for Map tasks.
- Allocate 28 containers on each node
- Use the default block size 64Mb.

Configuration parameters	Default value	Baseline Value
yarn.nodemanager.resource.memory-mb	8192	58296
yarn.nodemanager.resource.cpu-vcores	8	30
yarn.scheduler.minimum-allocation-mb	1024	2048
yarn.scheduler.maximum-allocation-mb	8192	58296
yarn.scheduler.minimum-allocation-vcores	1	1
yarn.app.mapreduce.am.resource.mb	1536	2048
yarn.app.mapreduce.am.command-opts	-Xmx1024m	-Xmx1624m
mapreduce.map.memory.mb	1024	2048
mapreduce.reduce.memory.mb	1024	2048
mapreduce.map.java.opts	-Xmx200m	-Xmx1624m
mapreduce.reduce.java.opts	-Xmx200m	-Xmx1624m
io.sort.mb	100	1000
io.sort.factor	10	100
mapreduce.reduce.shuffle.parallelcopies	5	20
dfs.block.size	64M	64M
mapreduce.map.output.compress	FALSE	FALSE
mapreduce.map.output.compress.codec	DefaultCodec	DefaultCodec
mapreduce.output.fileoutputformat.compress	FALSE	FALSE
mapreduce.output.fileoutputformat.compress.codec	DefaultCodec	DefaultCodec
mapreduce.job.reduce.slowstart.completedmaps	0.05	0.05
yarn.nodemanager.pmem-check-enabled	TRUE	TRUE

Table 5.1: Baseline Configuration

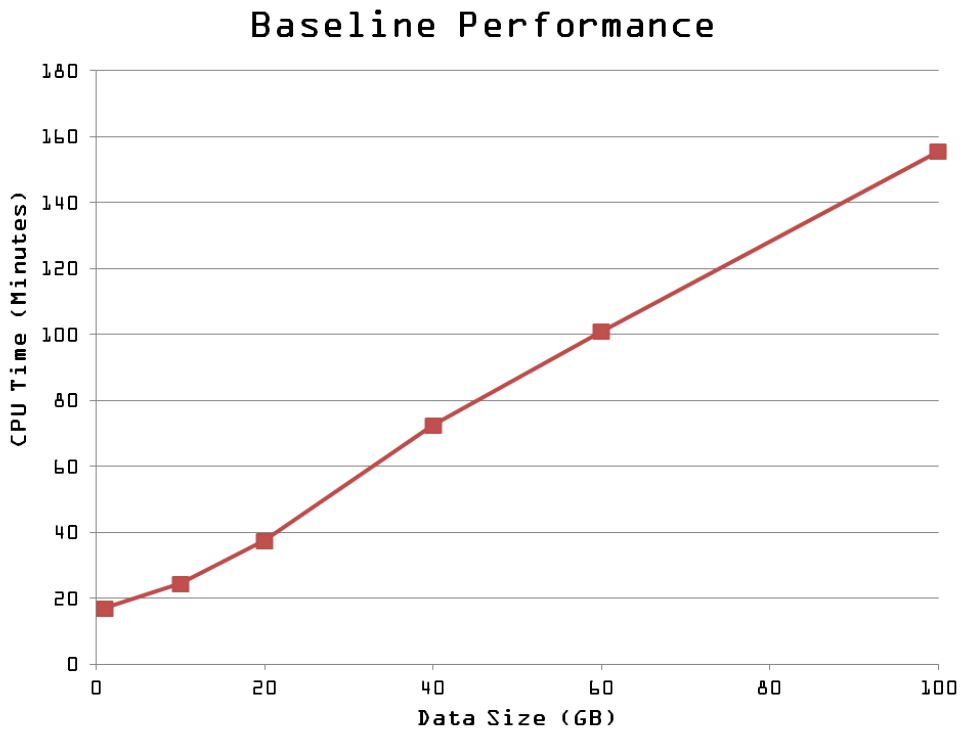

Figure 5.1: Baseline Performance Guided by General Tuning Practices

5.3.2 Baseline Result

Recall that each node in the cluster has 60GB RAM. YARN can only allocate up to 58296 MB of the amount to containers. The remaining memory is reserved for the OS kernel, system processes and other non-Hadoop processes. After running a bunch of jobs with various number of reducers, estimations of reducer numbers at various workloads are found and shown in Table 5.2.

Input Size(GB)	# of mappers	# of reducers
1	16	60
5	78	100
10	156	200
20	311	400
40	622	800
60	954	1200
100	1586	2000

Table 5.2: Numbers of Mappers and Reducers for Different Data Size

Using the baseline configuration and this set of reducer numbers, k-mer counting function runs against the 7 data sizes. The results are shown in Figure 5.1, which also illustrates the linear performance trend.

Then the counters and logs of these jobs were collected and analyzed. It turned out that the hardware is not fully untilized - there is room to improve the performance. In the next chapter the jobs are tuned from five perspectives.

Chapter 6

Performance Tuning

6.1 Challenges in Tuning Hadoop

The Hadoop community is growing exponentially as the amount of data generated by the internet and scientific research increases at unprecedented speed. The advantage of Hadoop over other parallel frameworks makes it adopted by many large companies and labs. The advantages include but not limited to:

- A simplified programming model
- Linear scalability to large number of nodes.
- Automatic data distribution and failure recovery

However one challenge with Hadoop is that tuning parameters is a rather time-consuming but necessary process. Factors, such as the implementation of map and reduce function, hardware resources and Hadoop configuration parameters, all affect the overall performance and especially the job execution time. Hadoop configuration parameters control the behaviors of jobs. It indicates the number of tasks that are allowed to run in parallel and specifies the runtime command line options of the job. Practice shows parameter tuning is a time consuming and daunting task due to the following reasons:

- Parameters cannot be tuned until the internals and interrelationship of Hadoop components are well understood.
- Hadoop is distributed with a bunch of configuration files and a very large number of parameters. The allocation and use of memory, CPU, I/O and network bandwidth are

all controlled by these files. Some parameters works collaboratively - changing value of one parameter may amplify or weaken the effect of another parameter.

- Default parameter values usually lead to underperformance due to default values are set for beginners, who usually start with single node.

- Due to the cluster resource limitations, setting values too aggressively may cause task failure.

Although there is lots of related research on this topic [32, 31, 15, 19, 18] there is no hard and fast set of rules to achieve optimal performance, because the input data size, workload characteristics and cluster hardware resource collaboratively decide the near-optimal values for these parameters. Also, tuning Hadoop performance is not just tuning one or more parameters, but getting balanced resources utilization.

Due to these difficulties, Hadoop performance tuning is an iterative process, including launching a job, analyzing Hadoop counters/logs, adjust parameters, and re-run the job. This process is repeated until the near-optimal performance reaches.

6.2 Configuration Difference between Hadoop 1 and Hadoop 2

As mentioned earlier, Hadoop 2 has undergone big architecture shift by adopting YARN as its resource manager. In Hadoop 2, parameters related to tasks are separated from mapred-site.xml into yarn-site.xml. Due to the different scheduling mechanism of Hadoop 2, some parameters are removed or depreciated and new container parameters are added. More precisely, YARN uses container as its allocation unit rather than slot in Hadoop 1.

The differences between Hadoop 1 and Hadoop 2 has been addressed comprehensively in a lot of literatures. The following only discusses two important ones pertinent to k-mer counting application: resource allocation and spill mechanism.

6.2.1 Resource Allocation: Slots vs. Containers

Hadoop 1: Slots

In Hadoop 1, the number of slots per node is set at the time that TaskTracker starts. It is controlled via the two parameters in mapred-site.xml: *mapred.map.tasks.maximum* and *mapred.reduce.tasks.maximum*. JobTracker lauches a JVM process for each map or reduce task, whose command line options can be passed by the *mapred.map.child.java.opts* and *mapred.reduce.child.java.opts*. Maximum heap size can be set by -Xmx option.

Hadoop 2: Containers

In Hadoop 2, YARN currently supports resource management for memory and CPU. The available resource on each Node Manager (NM) node in Hadoop cluster is divided into chunks and allocated as container - an encapsulation of resource elements (memory, CPU) and basic unit of processing capacity in YARN. YARN only allows a container to start if the node has enough resource to meet the container's requirement. Every process in Hadoop 2, including AppMaster, map task and reduce task, runs within containers.

The upper bound of physical RAM YARN can allocate on a node are specified by *yarn.nodemanager.resource.memory-mb* property in yarn-site.xml. This value should be set lower than the total memory available on each node because resources have to be reserved for system processes and other user processes. The memory size for a Map or Reduce task can be specified via *mapreduce.map.memory.mb* and *mapreduce.reduce.memory.mb* in mapred-site.xml. The JVM heap size is set via the property *mapreduce.map.java.opts* and *mapreduce.reduce.java.opts* and should be lower than the allocation size of containers. If memory usage of a task process grows beyond the container's setting (for example the task program has memory leak), the process will be killed by resource manager. By enforcing the resource limits, YARN ensures the cluster not being deteriorated by inexperienced users.

There is no pre-defined numbers of slots for Map or Reduce tasks in Hadoop 2. Instead, the total numbers of Hadoop processes including Map and Reduce tasks is limited by the number of containers, which is determined by the following formula:

```
min(
yarn.nodemanager.resource.memory-mb / mapreduce.[map|reduce].memory.mb,
yarn.nodemanager.resource.cpu-vcores / mapreduce.[map|reduce].cpu.vcores
)
```

Parameter Values

How to set these parameters is still hard even having understood the mechanism of resource allocation. There is a dilemma here. If starting more tasks simultaneously for a high utilization of CPU, the memory allocated for each task has to be less which may impair performance by spilling data on disks. On the other hand, if allocating more memory to task processes to increase the data throughput and decrease disk spills, the number of task processes has to be decreased, which makes the CPU underutilized.

The tradeoff requires careful calculation, evaluation and even testing. For our EC2 cluster the final setting is allocating 27 slots/containers for both Hadoop 1 (18 Map and 9 Reduce) and Hadoop 2. Setting the numbers to be same makes the comparison fair. The details of configuration is shown in Table 6.1

Configuration parameters	**Hadoop 2**	**Hadoop 1**
yarn.nodemanager.resource.memory-mb	55296	
yarn.nodemanager.resource.cpu-vcores	27	
mapreduce.map.memory.mb	2048	
mapreduce.reduce.memory.mb	2048	
mapred.map.tasks.maximum		18
mapred.reduce.tasks.maximum		9
# of concurrent tasks	27	27

Table 6.1: Hadoop Configuration

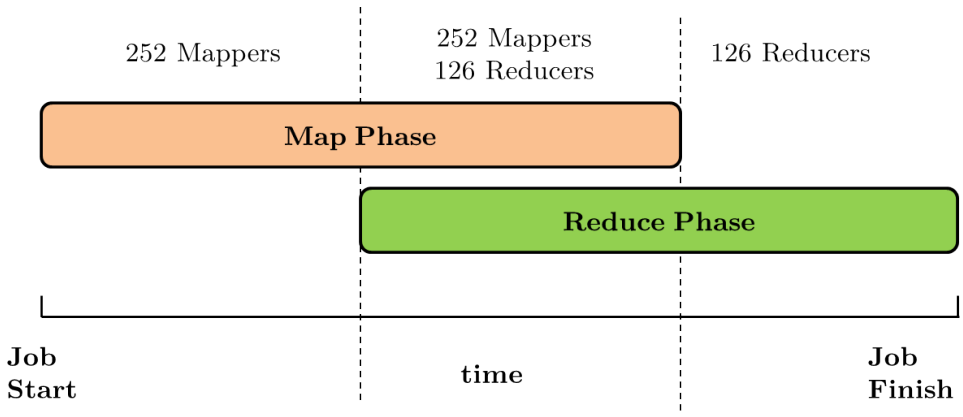

Figure 6.1: Concurrent Task Numbers in Hadoop 1

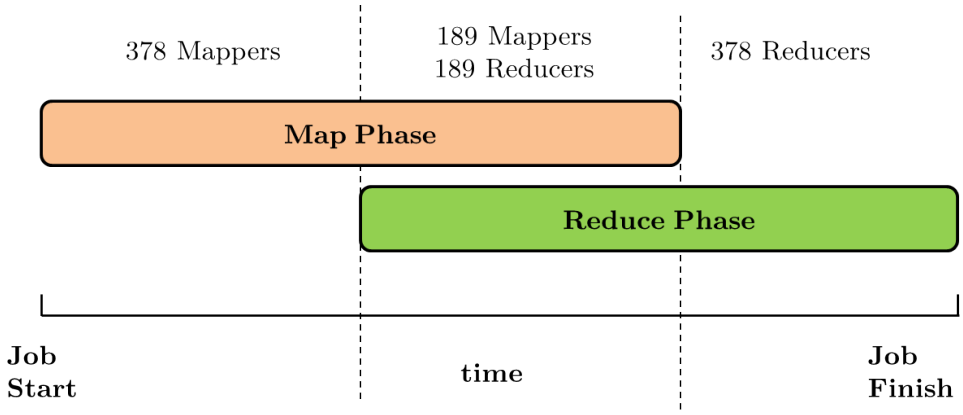

Figure 6.2: Concurrent Task Numbers in Hadoop 2

With this settings, the maximum number of task process on each node is roughly 27 (application master also takes a container in Hadoop 2). There are totally 14 data nodes in the cluster; hence the total number of slots/container is 14x27=378. However since slots are dedicated in Hadoop 1 while containers are shared in Hadoop 2, the number of concurrent tasks in Hadoop 1 is lower than the number in Hadoop 2. Figure 6.1 and Figure 6.2 decates this difference during the three running phases.

6.2.2 Map Side Spills

Hadoop 1

While a map task is executing, intermediate data is temporarily stored into a circular buffer which is a chunk of JVM heap space. The buffer is divided into two parts: one is

reserved for data records and another one is for metadata. The percentage of metadata in the buffer is governed by the *io.sort.record.percent* parameter in mapred-site.xml. By default the size of the sort buffer is 100MB and the percentage is 0.05. That is, 5MB in the buffer is reserved for the metadata and 95MB for the data records by default. The spilling is controlled by *io.sort.spill.percent* parameter in mapred-site.xml. When either the data or the metadata in the buffer reaches this percentage threshold, a background thread sorts and spills the contents of the buffer to disks. By default the percentage is equal to 0.8(80%). Practice shows metadata portion is saturated much faster than the data portion, which means even the data buffer is still far from full, a new spill is triggered by metadata threshold. To decrease the number of spills tuning of io.sort.record.percent manually is necessary, which is a tricky process.

Hadoop 2

In Hadoop 2 the io.sort.record.percent property was removed. Both the metadata and data records share the same space and they can vary in size. A spill would be triggered as soon as the whole size of records and metadata reaches the threshold. Hence less tuning is required to avoid over-spilling in Hadoop 2. This difference is summarized in Figure 6.3.

6.3 k-mer Counting Workload Characteristics

The near-optimal set of parameters for Hadoop depends on the data characteristics, application logics and hardware resources. Characteristics of intermediate data are vital to a MapReduce application. Table 6.2 shows the ratio of intermediate data size to the input data size for most common Hadoop applications. In contrast table 6.3 shows the ratio of k-mer counting application.

Notice that k-mer counting application generated much larger size of intermediate data compared to other applications. Taking 100GB input as an example, more than 1TB intermediate data are generated by map tasks. This distinctive characteristic makes most of

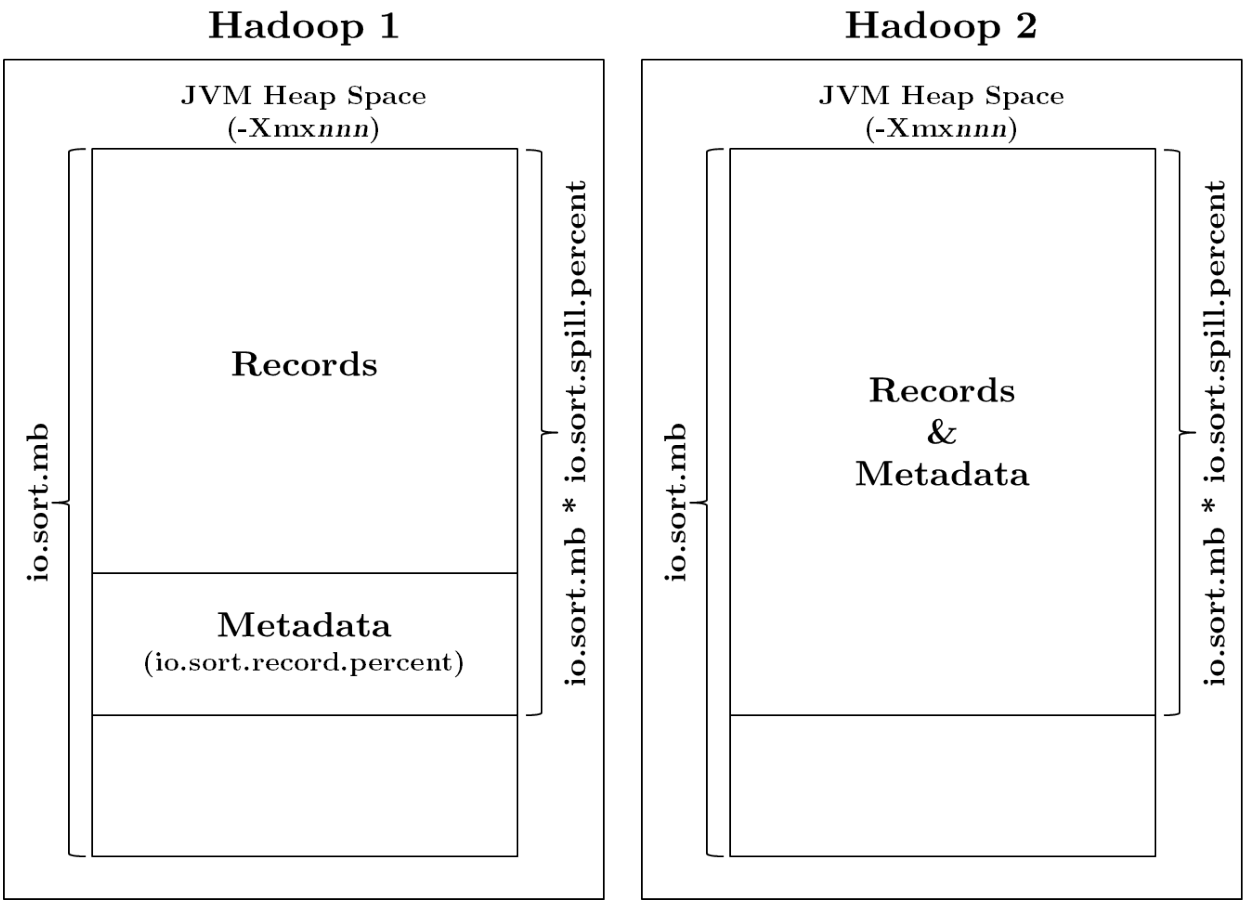

Figure 6.3: Comparison of Map-side Sort Buffer between Hadoop 1 and Hadoop 2

available Hadoop tuning guidelines inapplicable for k-mer counting application. According to its characteristics the following section tunes the parameters in 5 consecutive steps: data compression, HDFS blocks size, Map output spill, JVM garbage collection and reducer start-time. For each step, the performance of 40GB and 60GB workloads are compared to the performance of prior step. Testing data are provided to demonstrate the effect of each factor.

6.4 Parameters Tuning

6.4.1 Data Compression

k-mer counting application generates much larger volume of intermediate data than common applications. These intermediate data are stored on the local disk of the node where the task is executed. Hadoop job that generates a significant amount of map output

Job Name	Input size (TB)	Int. data size (TB)	Int./Input
LogProc	1.10	1.10	100%
NdayModel	3.54	3.54	100%
BehaviorModel	3.60	9.47	263%
ClickAttribution	6.80	8.20	121%
SegmentExploder	14.10	25.20	179%
LogRead	1.10	1.10	100%
LogCount	1.10	0.04	4%

Table 6.2: Characteristics of Intermediate Data for Common Hadoop Applications

Input size (GB)	Int. data size (GB)	Int./Input
1	13.5	1350%
5	67.7	1354%
10	135.4	1354%
20	270.9	1355%
40	541.5	1354%
60	830.0	1383%
100	1381.0	1381%

Table 6.3: Characteristics of Intermediate Data for k-mer Counting (k=20)

may benefit from the intermediate data compression without any application changes since compression reduces disk IO and the amount of data transferred via HTTP to reducers. Although compression and decompression may add extra load to the CPU, the overall job execution time can be decreased by enabling compression for the large volume of intermediate data.

Hadoop supports multiple compression formats (zlib, gzip, LZO, bzip2, Snappy etc). Snappy was chosen for the testing because it has a good balance between speed and space. By default the compression is disabled. To enable it follows the settings in Table 6.4

Parameter	Value
mapreduce.map.output.compress	true
mapreduce.map.output.compress.codec	org.apache.hadoop.io.compress.SnappyCodec
mapreduce.output.fileoutputformat.compress	true
mapreduce.output.fileoutputformat.compress.codec	org.apache.hadoop.io.compress.SnappyCodec

Table 6.4: Configuration of Data Compression

Tuning Result

The result in Table 6.5 shows data compression yielded more than 50% decrease in disk IO. Taking 40GB workload as an example, the *Number of bytes read* and *Number of bytes written* decreased from 604GB to 283GB and from 1200GB to 550GB respectively. Even though high speed SSD was used for EC2 cluster, the overall job runtime still decreased by 6 minutes for 40GB input and 8 minutes for 60GB input. It is not unrealistic to expected to decrease more in clusters with HDD storages.

Data Size	Counter Group	Uncompressed			Compressed			Diffrence		
		Map	Reduce	Total	Map	Reduce	Total	Map	Reduce	Total
40GB	Number of bytes read(GB)	604	598	1,202	283	131	414	321	467	788
	Number of bytes written(GB)	1,200	598	1,798	550	131	681	649	467	1,117
60GB	Number of bytes read(GB)	929	917	1,846	442	167	609	487	751	1,237
	Number of bytes written(GB)	1,839	917	2,756	853	167	1,020	986	751	1,736

Table 6.5: IO Improvement from Data Compression

6.4.2 HDFS Block Size

HDFS block size is one of the most well-known parameters, which is set by *dfs.block.size* in hdfs-site.xml. Data is split into blocks and the blocks are stored on data node. A map task is created for each block by default. Hence the block size and input data size of a Hadoop workload determine the number of Map tasks. Given an input file, the larger the block size is, the fewer Map tasks will be spawned in the Hadoop cluster. Almost all the available research relating to Hadoop tuning recommends a large block size. They claim that a large block size improves performance because small block size means large number of Map tasks and hence leads to more overhead of starting up and tearing down of Map JVMs. However k-mer counting is an exception due to its large volume of intermediate data. Analyzing the job logs found that large block size leads to more map-side spills. The overhead of merging the spills is more than the benefits of small block size. Given the JVM heap settings in the prior section, testing shows that a block size of 32MB gave the better performance than a

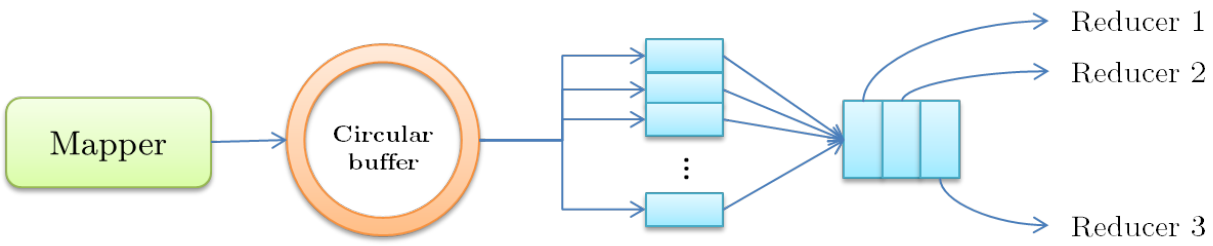

Figure 6.4: Shuffle and Sort Process

block size of 64MB. With this change the overall job execution time decrease by 12 minutes for 40GB input and by 17 minutes for 60 GB input.

6.4.3 Map Side Spill

A map task first writes its output to a circular buffer. Whenever the buffer reaches a certain threshold, the content of the buffer is sorted and spilled to local storage by a background thread. One map task may generate multiple spills depends on the buffer size and map output data size. If more than one spills happens, the spilled data have to be merged into a single sorted file partitioned by reduce keys. Reducer tasks pull their input from this partitioned merged file via HTTP. The details of this process is shown in the Figure 6.4.

Generally at least one spill happens for a map process. Additional spills during map phase make additional overhead of reading and merging of the spilled records. If *Map Output Records* counter is less than *Spilled Records* counter significantly, it means there are multiple spills. The map task logs also shows the spilling events with timestamps. With the logs the spill data size, the duration of spilling can be calculated. Three core parameters, which is shown in Table 6.6, in mapred-site.xml control the merge-sorting process .

Spill mechanism implies that the performance is best when the intermediate data is well contained in the sort buffer. Our analysis shows that even when the block size has been decreased, there is still more than 1 spill. Increasing *io.sort.mb* to 1100M and increasing

Parameter	Description	Default
mapreduce.task.io.sort.factor	The number of streams to merge at once while sorting files. This determines the number of open file handles	10
mapreduce.task.io.sort.mb	The total amount of buffer memory to use while sorting files, in megabytes. By default, gives each merge stream 1MB, which should minimize seeks.	100
mapreduce.map.sort.spill.percent	The soft limit in the serialization buffer. Once reached, a thread will begin to spill the contents to disk in the background. Note that collection will not block if this threshold is exceeded while a spill is already in progress, so spills may be larger than this threshold when it is set to less than .5	0.8

Table 6.6: Parameters Related to Map-side Spills

maximum heap size to 1724M eliminated the additional spills. Table 6.7 shows more than 50% of IO overhead was eliminated by this tuning.

Data Size	Counter Group	Spill			No spill			Diffrence		
		Map	Reduce	Total	Map	Reduce	Total	Map	Reduce	Total
40Gb	Number of bytes read	283	131	414	0	125	125	283	6	289
	Number of bytes written	550	131	681	278	125	403	272	6	278
	Map output records	32	0	32	32	0	32	0	0	0
	Spilled Records	63	31	95	32	32	63	31	0	31
60Gb	Number of bytes read	442	167	609	0	156	156	442	11	453
	Number of bytes written	853	167	1,020	431	156	587	422	11	433
	Map output records	49	0	49	49	0	49	0	0	0
	Spilled Records	97	48	145	48	48	97	48	0	48

Table 6.7: IO Reduction from Map-side Spill Tuning

6.4.4 JVM Garbage Collection

The core of Hadoop framework is written in Java programming language. Map and Reduce tasks run in their own JVM processes. Java garbage collection is an automatic process to manage the runtime memory used by JVM. Analyzing GC logs may reveal unreasonable JVM settings for an application. To enable GC logs, options of *-verbose:gc -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps* has to be appended to *mapreduce.map.java.opts* or *mapreduce.reduce.java.opts* parameters.

The following log shows the GC activities for a map process with max heap size of 2048M.

```
6.120: [GC [PSYoungGen: 242176K->19619K(282112K)] 242176K->19691K(926208K), 0.2651340 secs]
    [Times: user=0.29 sys=0.01, real=0.27 secs]
8.895: [GC [PSYoungGen: 261795K->8291K(524288K)] 1388267K->1134771K(1922048K), 0.3068380
    secs] [Times: user=0.34 sys=0.02, real=0.31 secs]
10.541: [GC [PSYoungGen: 492643K->6038K(524288K)] 1619123K->1132526K(1922048K), 0.2413280
    secs] [Times: user=0.23 sys=0.01, real=0.24 secs]
12.140: [GC [PSYoungGen: 490390K->6102K(659456K)] 1616878K->1132598K(2057216K), 0.3230520
    secs] [Times: user=0.34 sys=0.01, real=0.33 secs]
13.869: [GC [PSYoungGen: 625622K->5942K(625664K)] 1752118K->1132438K(2023424K), 0.1553950
    secs] [Times: user=0.23 sys=0.01, real=0.16 secs]
15.563: [GC [PSYoungGen: 625462K->6166K(641024K)] 1751958K->1132670K(2038784K), 0.2545130
    secs] [Times: user=0.27 sys=0.00, real=0.26 secs]
17.311: [GC [PSYoungGen: 626710K->128K(640000K)] 1753214K->1132468K(2037760K), 0.2842950
    secs] [Times: user=0.28 sys=0.00, real=0.29 secs]
20.868: [GC [PSYoungGen: 620672K->19960K(531968K)] 1753012K->1312716K(1929728K), 0.8527240
    secs] [Times: user=0.89 sys=0.09, real=0.85 secs]
21.721: [Full GC [PSYoungGen: 19960K->0K(531968K)] [ParOldGen: 1292756K->1312033K(1397760K)]
     1312716K->1312033K(1929728K) [PSPermGen: 28974K->28956K(58368K)], 45.0386940 secs] [
    Times: user=73.50 sys=1.13, real=45.04 secs]
70.555: [Full GC [PSYoungGen: 512000K->0K(531968K)] [ParOldGen: 1312033K->1133327K(1397760K)
    ] 1824033K->1133327K(1929728K) [PSPermGen: 29102K->29101K(65536K)], 0.3149510 secs] [
    Times: user=0.21 sys=0.01, real=0.32 secs]
```

There are two types of GC shown in the log: *GC* (aka Minor GC) and *Full GC*. Minor GC is a lightweight operation which only takes less than one second because it only cleans the *Yong Generation*. Full GC is a heavy one which cleans the full heap space including Yong Generation, *Old Generation* and *Permanent Generation*. The log shows that there is one Full GC that takes 45 seconds, which implies that GC tuning is necessary.

It turns out the Full GC is triggered by expanding the Permanent Generation, which is caused by the small initial setting (default setting). Based on guidelines in the paper [30], Permanent Generation is set to 128Mb and 4 parallel threads are set to perform the garbage collection.

Table 6.8 shows the performance improvement from the tuning. The overall job running time is decreased by 9 minutes for 40GB input and 11 minutes for 60GB input.

Data Size	Counter Group	before GC tuning			After GC tuning			Diffrence		
	GC Time (s)	Map	Reduce	Total	Map	Reduce	Total	Map	Reduce	Total
40GB	Job1	183,905	529	184,433	2,014	104	2,118	181,891	424	182,315
	Job2	18,807	5,736	24,543	7,720	1,510	9,230	11,087	4,226	15,313
60GB	Job1	25,283	390	25,672	7,135	141	7,277	18,147	249	18,396
	Job2	26,577	8,120	34,697	11,040	2,302	13,342	15,538	5,818	21,355

Table 6.8: Performance Gain from GC Tuning

6.4.5 Reducer Start Time

The timeline of a MapReduce job can be divided into Map phase and Reduce phase. Map phase doesn't need large network bandwidth because data locality guarantees map tasks running on the nodes where the input data store and map output is only written into local disks. In contrast, the reduce phase, which gathers and combines the output from all the mappers, is network IO intensive, because each reducer pulls its input from almost every other nodes and writes its output into HDFS. More precisely the Reduce phase can be divided into 3 parts:

- Shuffle: collects input from mappers:
- Sort: sorts the records and merges them by keys.
- Reduce: runs the reduce program, then write its result into HDFS.

Figure 6.5 shows the timeline of a MapReduce job execution. Notice that shuffle may start before the end of the Map phase but only finish after all map tasks have finished. Sort and reduce may only start when the shuffle completes.

Since the map and shuffle overlaps, coordination of them is crucial for the overall job execution time. The time that a reducer begins shuffle is controlled by *mapred.reduce.slowstart.completed.maps* parameter (default 5%). Shuffle only starts when the progress of the map task exceeds the parameter value. When there are a lot of map

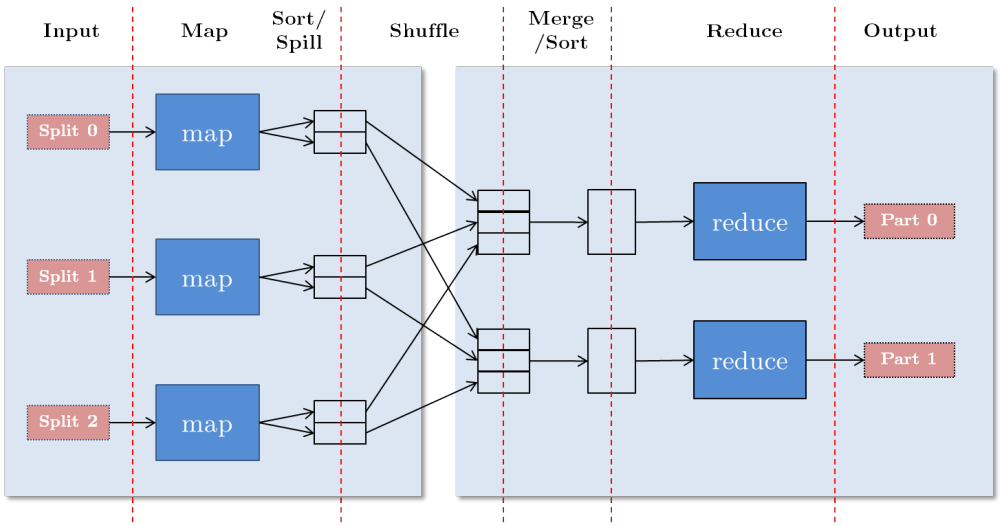

Figure 6.5: Decomposition of Reducer Phases

tasks performance suffers for the low default value. The started reducer processes get stuck at the shuffle phase to wait for the completion of each map task, which is a waste of process resource since the total number of containers is fixed. After having tried a set of different values, mapred.reduce.slowstart.completed.maps parameter was set to 0.90 for the k-mer counting application. The overall job execution time decreased by 4 minutes for 40GB input and 8 minutes for 60GB input.

6.5 Tuning Results and Remarks

6.5.1 Tuning Results

Figure 6.6 and Figure 6.7 show how each tuning step affects the overall run time in details. Note that block size and JVM GC are two biggest factors and performance improvement roughly proportional to the data size.

Figure 6.8 shows the performance trend before and after tuning. The overall job execution time reduces by 44% for the 40GB input and 47% for the 60GB input. The linear scalability is kept.

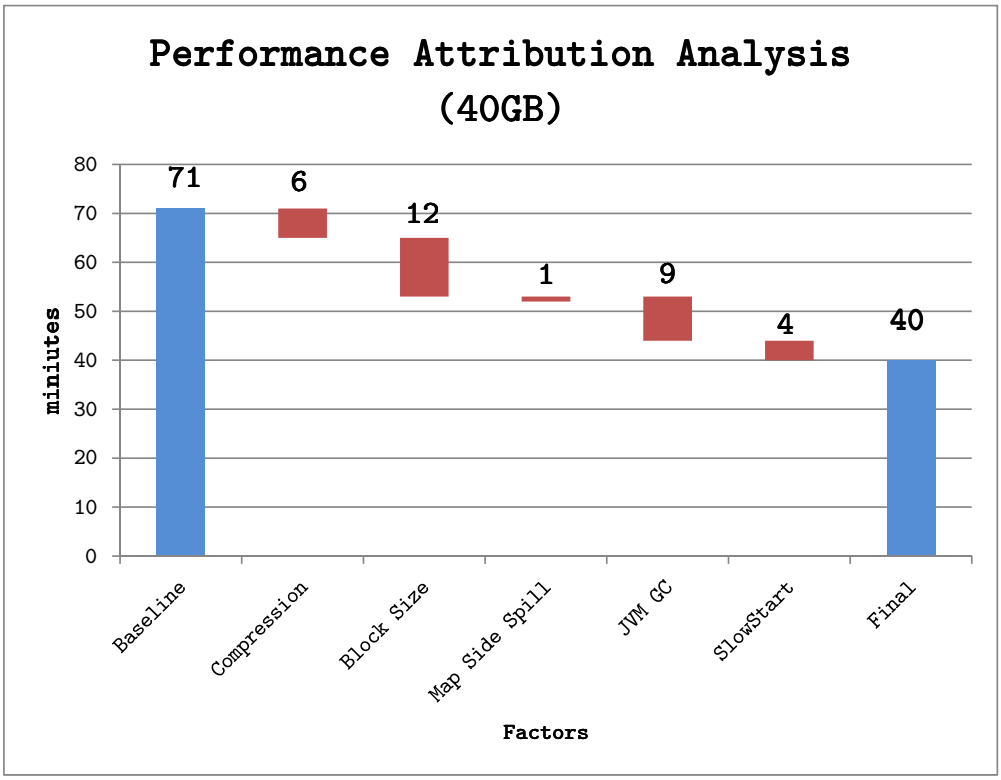

Figure 6.6: Impact Factors on 40GB

6.5.2 Remarks

Disk IO and network bandwidth are usually two performance bottlenecks for Hadoop applications. The configuration of SSD and 10Gigabit enhanced Ethernet for our EC2 cluster ease the burden of these bottlenecks. SSD delivers up to 70% higher MapReduce performance compared to HDDs of equal aggregate IO bandwidth [15] and 10 gigabit Ethernet allows data speeds up to 10 billion bits per second. So disk IO and bandwidth may not be the constraints to the overall performance in our case. We believes that applying these tunings to Hadoop clusters with HDD and general speed Ethernet [14] may bring much larger performance improvement.

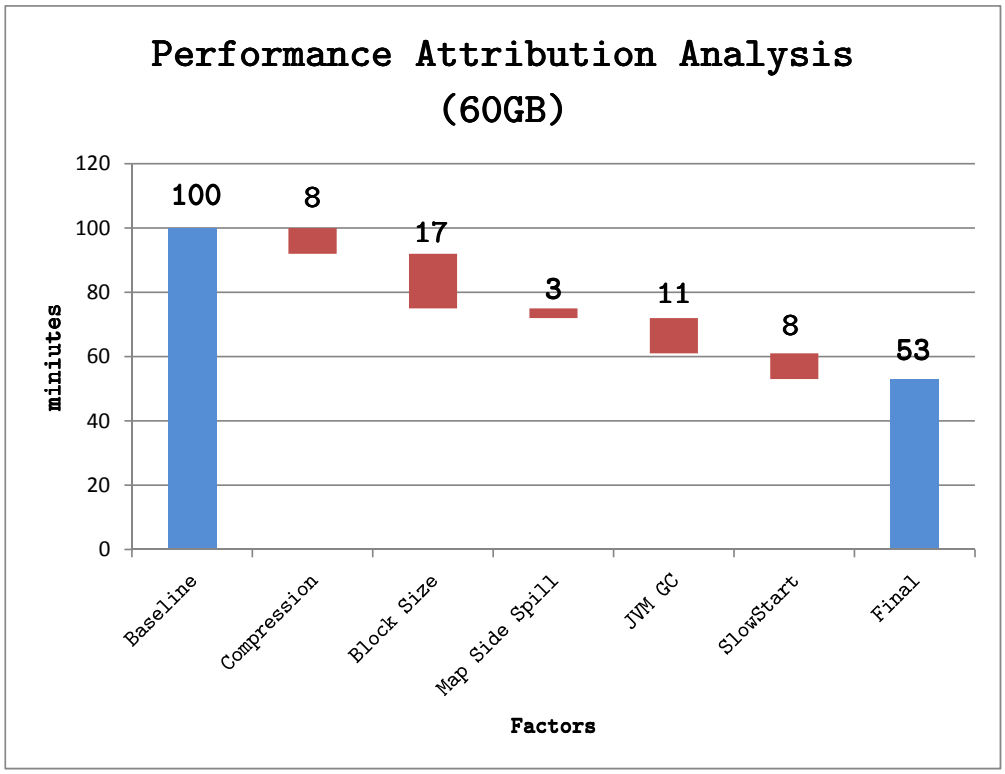

Figure 6.7: Impact Factors on 60GB

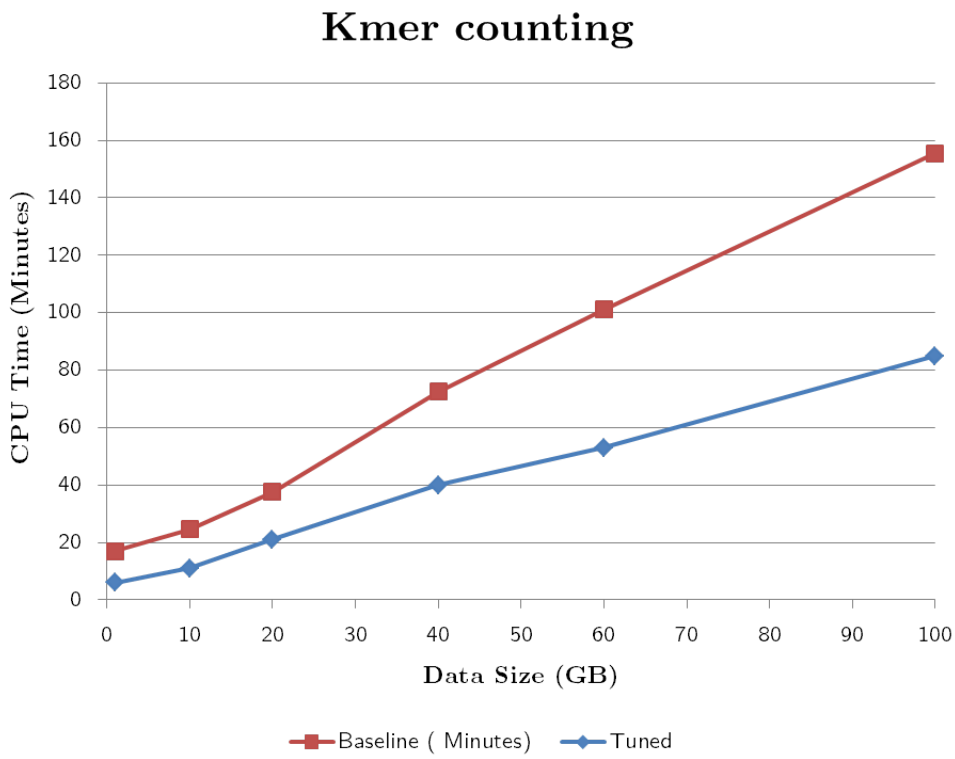

Figure 6.8: Performance Comparison

Chapter 7

Future Work

Even though parameter tuning can bring obvious performance improvement, IO operation is still the bottleneck of Hadoop-based application. we can further optimize BioPig job performance from the following 3 perspectives:

- using existing libraries or techniques, such as Hadoop-BAM [24], Apache Parquet [3] and Apache Avro [1], to further decrease disk IO overhead.
- implementing a combiner to reduce the amount of data to be transferred to the reducers is another optimization for the MapReduce job.
- migrating BioPig from Hadoop to Spark [4], which is an in-memory computing framework suitable for iterative and interactive applications. It is claims Spark may bring up to 100x faster than Hadoop framework.

Another issue is that even though AMI is created to simplify BioPig setup, it still seems to be complicate to use for researchers of Biology background. The ideal scenario is to provide web interfaces, just like what CloudMan [6] did for Galaxy. With web interfaces, users only need to specify parameters to launch a BioPig cluster.

Last but not the least, YARN-based BioPig currently includes 5 modules. More Bioinformatics applications may be added to the toolkit to enhance the functionality of BioPig.

Chapter 8

Conclusion

The emergence of massive datasets in Bioinformatics presents challenges in sequence analysis. Hadoop MapReduce framework, which was designed to get its parallelism from large collections of commodity hardware, is adopted to address the challenges. Currently, a bunch of MapReduce-based Bioinformatics tools are available on the market. BioPig is one of them. In this thesis, I introduced YARN-based BioPig toolkit for large-scale sequence analysis. With YARN, the power of cluster computing with Hadoop was enhanced. Job throughput and cluster utilization were improved.

To further improve YARN-based BioPig performance, I tuned Hadoop parameters from 5 perspectives according to k-mer counting characteristics. Result shows these tuning achieved an average performance improvement of about 50% compared to baseline configuration. Aside from migration and tuning work, scalable sequence distance functions were implemented to extend YARN-based BioPig framework.

Bibliography

[1] Apache Avro. https://avro.apache.org/.

[2] Apache Hadoop. http://hadoop.apache.org/.

[3] Apache Parquet. https://parquet.apache.org/.

[4] Apache Saprk. http://spark.apache.org/.

[5] AWS. http://aws.amazon.com/.

[6] CloudMan. https://wiki.galaxyproject.org/CloudMan.

[7] EC2. http://aws.amazon.com/ec2/.

[8] Hadoop Yarn. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[9] Sequence Analysis. https://en.wikipedia.org/wiki/Sequence_analysis.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. Halvade: scalable sequence analysis with mapreduce. *Bioinformatics*, page btv179, 2015.

[12] J. Goecks, A. Nekrutenko, J. Taylor, et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[13] D. Hong, A. Rhie, S.-S. Park, J. Lee, Y. S. Ju, S. Kim, S.-B. Yu, T. Bleazard, H.-S. Park, H. Rhee, et al. Fx: an rna-seq analysis tool on the cloud. *Bioinformatics*, 28(5):721–723, 2012.

[14] Intel. Optimizing hadoop deployments.

[15] K. Kambatla and Y. Chen. The truth about mapreduce performance on ssds. In *Proc. USENIX LISA*, 2014.

[16] B. Langmead, K. D. Hansen, J. T. Leek, et al. Cloud-scale rna-sequencing differential expression analysis with myrna. *Genome Biol*, 11(8):R83, 2010.

[17] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for snps with cloud computing. *Genome Biol*, 10(11):R134, 2009.

[18] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 165–176. ACM, 2014.

[19] M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint optimization of overlapping phases in mapreduce. *Performance Evaluation*, 70(10):720–735, 2013.

[20] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. Adam: Genomics formats and processing patterns for cloud scale computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-207*, 2013.

[21] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 222–229. IEEE, 2008.

[22] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[23] T. Nguyen, W. Shi, and D. Ruden. Cloudaligner: A fast and full-featured mapreduce based tool for sequence mapping. *BMC research notes*, 4(1):171, 2011.

[24] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemelä, E. Korpelainen, and K. Heljanko. Hadoop-bam: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28(6):876–877, 2012.

[25] H. Nordberg, K. Bhatia, K. Wang, and Z. Wang. Biopig: a hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, page btt528, 2013.

[26] 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*, pages 1099–1110. ACM, 2008.

[27] L. Pireddu, S. Leo, and G. Zanetti. Seal: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, 27(15):2159–2160, 2011.

[28] M. C. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[29] A. Schumacher, L. Pireddu, M. Niemenmaa, A. Kallio, E. Korpelainen, G. Zanetti, and K. Heljanko. Seqpig: simple and scalable scripting for large sequencing data sets in hadoop. *Bioinformatics*, 30(1):119–120, 2014.

[30] V. L. Shrinivas Joshi. Java garbage collection characteristics and tuning guidelines for apache hadoop terasort workload. 2012.

[31] P. TUNING. Performance tuning. 2009.

[32] J. Varia. Architecting for the cloud: Best practices. *Amazon Web Services*, 2010.

[33] M. S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M. J. Okoniewski. Sparkseq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, page btu343, 2014.

[34] J. Zhang, R. Chiodini, A. Badr, and G. Zhang. The impact of next-generation sequencing on genomics. *Journal of genetics and genomics*, 38(3):95–109, 2011.