A Case Study of Tuning MapReduce for Efficient Bioinformatics in the Cloud

Lizhen Shi[a], Zhong Wang[b], Weikuan Yu[a], Xiandong Meng[b]

[a]*Florida State University, 600 W College Ave, Tallahassee, FL 32306*
[b]*Genomics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*

Abstract

The combination of the Hadoop MapReduce programming model and cloud computing allows biological scientists to analyze next-generation sequencing (NGS) data in a timely and cost-effective manner. Cloud computing platforms remove the burden of IT facility procurement and management from end users and provide ease of access to Hadoop clusters. However, biological scientists are still expected to choose appropriate Hadoop parameters for running their jobs. More importantly, the available Hadoop tuning guidelines are either obsolete or too general to capture the particular characteristics of bioinformatics applications. In this study, we aim to minimize the cloud computing cost spent on bioinformatics data analysis by optimizing the extracted significant Hadoop parameters. When using MapReduce-based bioinformatics tools in the cloud, the default settings often lead to resource underutilization and wasteful expenses. We choose k-mer counting, a representative application used in a large number of NGS data analysis tools, as our study case. Experimental results show that, with the fine-tuned parameters, we achieve a total of 4x speedup compared with the original performance (using the default settings). This paper presents an exemplary case for tuning MapReduce-based bioinformatics applications in the cloud, and documents the key parameters that could lead to significant performance benefits.

Keywords: Hadoop, YARN, Parameter Optimization, K-mer Counting, NGS

1. Introduction

To date, next-generation sequencing (NGS) has become the leading application area in the domain of bioinformatics. With the advent of NGS technologies, sequence datasets are growing exponentially and an ultra large-scale of data generation can be achieved. In current days, modern Illumina systems can generate hundreds of gigabytes of sequences per run with 99.9% accuracy [1]. Meanwhile, extremely large-scale sequencing projects are emerging, such as ENCODE project [2], 1000 Genomes project [3], Cow Rumen Deep Metagenomes project [4] and Human Microbiome project [5]. With the advent of these large-scale sequencing projects, traditional analysis tools have been challenged by the need to scale commensurately with the exponential growth of data size.

The Hadoop framework [6], Apache's open source implementation of Google's MapReduce programming model [7], has emeged as a viable solution to the challenge of processing an unprecedented amount of sequence data. The Hadoop Distributed File System (HDFS) and MapReduce are two core components of the Hadoop framework. With the assistance of HDFS, Apache Hadoop not only enables distributed, scalable, and fault tolerance data storage, but it also enables built-in data locality and distributed data processing. MapReduce permits tasks to run in a massively parallel manner on a large number of nodes. With the combination of MapReduce and HDFS, Hadoop provides a load-balanced, scalable, and reliable framework. Biological scientists have already harnessed the power of Hadoop in various scenarios; such as mapping NGS data to a reference genome, finding SNPs from short read data, and matching strings in genotype files. Table 1 concludes some of the available MapReduce-based bioinformatics tools on the market.

The accelerated adoption of the Hadoop MapReduce programming model in the domain of bioinformatics is due not only to the exponential growth of sequence data, but also to the popularity of cloud computing. This trend is anticipated to continue growing in the foreseeable future. Cloud computing removes the burden of hardware and software setup from end users and provides ease of access to Hadoop clusters. With the emergence of cloud computing, companies are attracted to move their data analytics tasks to the cloud due to its exible, on demand resources usage and pay-as-you-go pricing model. Amazon Web Services (AWS) [8] is the most popular cloud-computing platform offered by Amazon. With the assistance of customized Amazon Machine Images (AMIs), biological scientists can easily reproduce the computation environment of a particular application and its past results.

However, one issue with efficiently leveraging Hadoop is

Email addresses: lshi@cs.fsu.edu (Lizhen Shi), zhongwang@lbl.gov (Zhong Wang), yuw@cs.fsu.edu (Weikuan Yu), xiandongmeng@lbl.gov (Xiandong Meng)

Name	Description
CloudBLAST [9]	Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications
CloudBurst [10]	Highly sensitive read mapping with MapReduce
Biodoop [11]	Bioinformatics on Hadoop
Crossbow [12]	Searching for SNPs with cloud computing
GATK [13]	A MapReduce framework for analyzing next-generation DNA sequencing data
Myrna [14]	Cloud-scale RNA-sequencing differentialexpression analysis
Galaxy [15]	A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences
SEAL [16]	A Distributed Short Read Mapping and Duplicate Removal Tool
CloudAligner [17]	A fast and full-featured MapReduce based tool for sequence mapping
Contrail [18]	A de Bruijn Genome Assembler that uses Hadoop
FX [19]	an RNA-Seq analysis tool on the cloud
BioPig [20]	A Hadoop-based analytic toolkit for large-scale sequence data
SeqPig [21]	Simple and scalable scripting for large sequencing data sets in Hadoop
Halvade [22]	Scalable sequence analysis with MapReduce

Table 1: Available MapReduce-based Bioinformatics Tools

its large number of configuration parameters. The Hadoop default settings often lead to cluster underutilization and poor performance, resulting in expensive computation costs in the cloud environment. On the other hand, tuning these parameters requires an advanced knowledge of Hadoop, which could prevent biological scientists from using currently available (and useful) MapReduce–based tools for genomic analysis. Although there has been a lot of research on this topic [23, 24, 25, 26, 27, 28, 29], they all focus on common Hadoop applications, which have different characteristics from bioinformatics applications. The near-optimal configuration of Hadoop parameters is tightly coupled with application characteristics, so it is essential to tune MapReduce-based bioinformatics tools according to their specific characteristics.

In order to provide biological scientists a clear direction on how to minimize incurred computation costs by optimizing Hadoop parameters, in this paper, we first extract about 20 significant parameters from the Hadoop parameter space (including more than 200 parameters). These extracted parameters are classified into four groups: CPU relevant, memory relevant, disk relevant, and network relevant parameters. We then use k-mer counting in BioPig [20] as our study case to explore its characteristics and then identify and optimize the Hadoop parameters that are critical to its performance. K-mer counting is a typical application in bioinformatics and is used in various scenarios of NGS data analysis, including sequence quality assessment, read alignment, fast similarity search, short read assembly, taxonomy classification, etc. Based on an initial tuning configuration (in section 4), we conduct a fine-tuning consisting of 4 steps to further improve cluster utilization according to the k-mer counting characteristics. With our fine-tuned parameters, we show a performance improvement of about 50% when compared to the initial setup, and a total of 4x speedup compared to the Hadoop's default settings (this is detailed in section 4 and 5). We believe that this tuning experience can provide a valuable reference not only for bioinformatics applicaions, but also for applications with similar characteristics.

The paper is organized as follows. Section 2 introduces the background of Hadoop, BioPig, and k-mer counting. Section 3 provides the significant Hadoop parameters and their classification. Section 4 sets up an initial tuning for the k-mer counting, followed by the fine-tuning in Section 5. Section 6 gives a review of related work. Section 7 concludes the paper and discusses some future directions to further optimize k-mer counting performance.

2. Background

The wild adoption of the Hadoop framework and new types of applications push the original monolithic architecture (Hadoop 1) to the next generation of the Hadoop computing platform (Hadoop 2), in which YARN becomes a separate layer and acts as the resource manager of the Hadoop cluster. Most of the previous research on Hadoop tuning works on Hadoop 1 [23, 24, 26, 27, 28, 29]. This paper targets the latest version, Hadoop 2. In this section, we first give a general overview of the Hadoop framework. We next introduce DNA sequence analysis, BioPig toolkit, and K-mer counting.

2.1. Overview of the Hadoop Framework

Apache Hadoop as an open source implementation of the MapReduce paradigm provides a reliable, scalable, and distributed computing framework for data-intensive applications. Hadoop Distributed File System (HDFS) and Hadoop MapReduce are two integral components inside Hadoop.

2.1.1. Hadoop Distributed File System (HDFS)

Hadoop applications read data from and write data to HDFS. HDFS includes two types of nodes: one NameNode and many DataNodes. Data are broken into blocks

and distributed among all the DataNodes. Hardware failure is fairly common in Hadoop clusters in that Hadoop runs on commodity hardware. To achieve high availability, each block of data is replicated to 3 DataNodes, preventing the failure of one node from causing the loss of valuable data. Heavy network traffic is often one of the major bottlenecks for parallel and distributed computing. Having map task running on the node where its input data resides efficiently addresses this issue. This technique, known as data locality, is one of the key advantages of Hadoop over other parallel frameworks.

Even though the NameNode does not hold any dataset, it holds the metadata for the Hadoop cluster and monitors the health of the DataNodes via heartbeats from DataNodes. As a result the NameNode is the most critical component of the HDFS. Prior to Hadoop 2, the Hadoop cluster size is limited in that only one NameNode is allowed for the entire cluster. HDFS federation, an important feature in Hadoop 2, is designed to overcome the limitation of the single NameNode. Cluster storage can scale horizontally with the assistance of HDFS federation.

2.1.2. Hadoop MapReduce

MapReduce [7] provides a programming model for data-intensive processing in a massively parallel fashion. In MapReduce, the workload is decomposed into a large number of small tasks and distributed to a large number of nodes. Many real-world applications fit very well to the MapReduce paradigm and can be executed on MapReduce-based platforms.

The second generation of the Hadoop MapReduce, which we call MapReduce 2 (MRv2) or YARN [30], brings some benefits over its previous version. First, instead of dividing resources into map and reduce slots, it provides a unified resource unit called container, which provides a great deal of flexibility for dynamic resource allocation. Second, it relieves the single bottlenecked JobTracker through a hierarchical management scheme, in which a global ResourceManager is responsible for the coordination between applications within a Hadoop cluster and the per-application ApplicationMaster manages all tasks within a given application. Finally, YARN, as a seperate resource management layer, supports both MapReduce and non-MapReduce applications running on the Hadoop cluster.

2.2. DNA Sequence Analysis and BioPig Toolkit

DNA sequences consist of four unique bases labeled A, T, C, and G (for adenine, thymine, cytosine, and guanine respectively). Genomic analysis is the process of analyzing an organism's sequences (such as DNA) to understand its features, functions, structure, or evolution. Methodologies used include sequence alignment, searches against biological databases, etc [31, 32].

BioPig [20] is a Hadoop-based toolkit for large-scale DNA sequence analysis in bioinformatics. It is fully open source under the BSD license, and is implemented on top of Apache Hadoop framework and Pig [33] data flow language. Leveraging the advantages of the Hadoop and Pig frameworks, BioPig has shown its scalability, programmability and portability. BioPig has evolved into its second generation built on Hadoop 2 and Pig 0.15. BioPig consists of five main functional modules: Input/Output, K-mer counting, Blast, Assembly and Similarity.

2.3. K-mer Counting

K-mer counting is the core module in BioPig toolkit and a prerequisite step of many bioinformatics applications. K-mers refer to all the possible subsequences of length k in a DNA/RNA sequencing. Counting the occurrences of every k-mer in a genome sequence is the preliminary and central step of many subsequent analyses, such as constructing de Bruijn graphs [34] in sequence assembly, eliminating erroneous reads in a relatively large number of datasets and aligning multiple sequences. 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. Counting large k-mers of large modern sequence datasets can easily overwhelm the memory capacity of standard computers. To address this issue, BioPig framework provides a scalable k-mer counter which scales well with the dataset and k-mer sizes due to the linear scalability of Hadoop framework.

3. Significant Parameters Classification

Hadoop parameters are crucial to the resource allocation and job execution behavior. Hadoop exposes over 200 parameters that have varying impact on job performance, where we identify 20 significant ones based on our knowledge of the configuration parameters and our previous experience with Hadoop tuning. It is important to note that these parameters play a significant role in the performance of applications in the domain of, including, but not limited to, bioinformatics. We classify the extracted significant ones into 4 different groups (Table 2).

CPU Relevant Parameters

CPU is a first class resource from a scheduling perspective; its use governs the job performance fundamentally. In Hadoop, vcores, the abbreviation for virtual cores, are used for resource and container allocation, one vcore corresponds to one physical processor.

There are several CPU relevant parameters in Hadoop configuration files. To be specific, the *nodemanager.resource.cpu–vcores* specifies the number of vcores that can be allocated for containers on a single node. The *mapreduce.map|reduce.cpu.vcores* represents the number of vcores allocated for each map/reduce task. In YARN, the *yarn.app.mapreduce.am.resource.cpu–vcores* stands for the number of vcores the ApplicationMaster needs. The *yarn.scheduler.minimum|maximum-allocation–vcores* is the minimum/maxi

CPU	yarn.nodemanager.resource.cpu-vcores
	yarn.scheduler.maximum-allocation-vcores
Memory	yarn.nodemanager.resource.memory-mb
	yarn.scheduler.minimum-allocation-mb
	yarn.scheduler.maximum-allocation-mb
	yarn.app.mapreduce.am.resource.mb
	yarn.app.mapreduce.am.command-opts
	mapreduce.map.memory.mb
	mapreduce.reduce.memory.mb
	mapreduce.map.java.opts
	mapreduce.reduce.java.opts
Disk	io.sort.mb
	io.sort.spill.percent
	dfs.block.size
	mapreduce.map.output.compress
	mapreduce.map.output.compress.codec
	mapreduce.output.fileoutputformat.compress
	mapreduce.output.fileoutputformat.compress.codec
Network	mapreduce.job.reduce.slowstart.completedmaps
	mapreduce.reduce.shuffle.parallelcopies

Table 2: Significant Parameters Classification

mum allocation for each container request at the YARN ResourceManager.

For the ApplicationMaster, map, and reduce tasks, they typically run in a single thread and use one vcore, which is the same as the default settings. However, the total number of vcores available on each node varies from cluster to cluster, we need to reconfigure the *yarn.nodemanager.resource.cpu-vcores* and *yarn.scheduler.maximum-allocation-vcores* on a given cluster. The *yarn.nodemanager.resource.cpu-vcores* should be set lower than the total number of vcores available on each node because some resource have to be reserved for the HDFS DataNode and the YARN NodeManager, as well as other non-MapReduce processes. At the same time, the *yarn.scheduler.maximum–allocation–vcores* should be set greater than *yarn.scheduler.minimum-allocation-mb* and no more than *yarn.nodemanager.resource.cpu–vcores*. Tuning these two CPU relevant parameters often suffices because most MapReduce applications are I/O bound. What users usually look into is to optimize memory, disk, and network usage, which will be discussed in the next.

Memory Relevant Parameters

In Hadoop 2, container as the unified allocation unit provides a great deal of flexibility for dynamic adaption. YARN only allows a container to start if a node has enough resource to meet the container's requirement. Currently, container encapsulates memory and CPU, most of its requirements are specified using memory. More precisely, the memory relevant parameters can be classified into two categories.

1) YARN

YARN can allocate an upper bound of memory on a node using the *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 some resources have to be reserved for system processes, as well as other user processes. The minimum/maximum amount of memory for each container request is determined by *yarn.scheduler.{minimum|maximum}-allocation-mb*. ApplicationMaster is the application's first container launched by YARN ResourceManager, its memory requirement and JVM heap size are specified by *yarn.app.mapreduce.am.resource.mb* and *yarn.app.mapreduce.am.command-opts* respectively.

2) MapReduce

Both the map task and reduce task run in containers. The memory size of a container for each map/reduce task can be specified via *mapreduce.{map|reduce}.memory.mb* in mapred-site.xml. The JVM heap size of the map/reduce container is governed by *mapreduce.{map|reduce}.java.opts* and should be set slightly lower than the corresponding container size.

In addition, Hadoop is implemented in Java. Garbage Collection (GC) allows Java programmer to focus on solving their logic specific problems without managing memory. However, it may become a significant performance bottleneck. Analyzing GC logs may reveal unreasonable GC settings for an application. To enable GC logs, options of *-verbose:gc*, *-XX:+PrintGC*, *-XX:+PrintGCDetails* and *-XX:+PrintGC TimeStamps* have to be appended to the existing *JAVA_OPTS*. Among the above memory relevant parameters, JVM options can help tune GC behavior.

Disk Relevant Parameters

With Hadoop framework, not only are the input and output stored in disk (HDFS), but so is the intermediate data. I/O is often the bottleneck that constrains the Hadoop application performance. Data compression and spilled records reduction are often two effective ways to reduce I/O operations. Data compression allows the data can be stored in more compact form. Spilling occurs frequently in the map side due to its limited sorting buffer size. To be specific, a map task first writes its output to a circular buffer, which size is controlled by *io.sort.mb* property. Whenever the buffer reaches a certain threshold governed by *io.sort.spill.percent* parameter, the content of the buffer is sorted and spilled to local storage by a background thread. One map task may generate multiple spills depending on the buffer size and map output data size. Spill mechanism implies that the performance is best when the spilled records is identical to the number of map output records, which means that map output is just spilled once. If more than one spills are generated, the spilled records have to be reread and rewritten into a single sorted file partitioned by reduce keys, incurring the additional overhead of disk I/O. The detail of this process is shown in figure 1.

dfs.block.size is another important disk relevant parameter. Because one map task is generated for each HDFS block, decreasing the HDFS block size can reduce the data map task need to process, meaning less output would be generated.

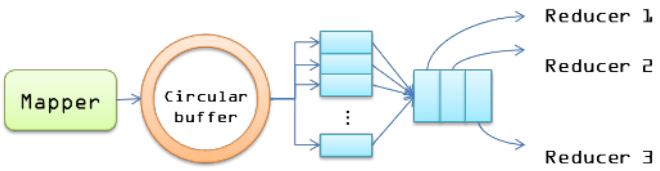


Figure 1: The Flow of Data Processing across MapReduce Tasks

To sum up, data compression and spilled records reduction are two efficient ways to reduce the I/O overhead. With regard to the spilled records reduction, there are also two different ways: decreasing the HDFS block size to reduce the map output and increasing the sorting buffer size to fit the map output.

Network Relevant Parameters

The lifetime of a MapReduce job can be divided into two phases: map and reduce. The map phase doesn't require high network bandwidth because the scheduling locality of map tasks helps co-locate these tasks where the input data is stored. Map output is only written to local disks. In contrast, the reduce phase, which gathers and combines the output from map phase, incurs heavy network traffic in that each reduce task pulls its input from every map task accross the network and writes its output into HDFS. As a result, improving network traffic condition and preventing traffic congestion are crucial for the overall job performance.

More precisely, the reduce phase can be divided into 3 steps:

Shuffle: Collects input from map tasks.

Sort: Sorts and merges the records by keys.

Reduce: Runs the reduce function and writes its output into HDFS.

Figure 2 shows the timeline of a MapReduce job execution. Shuffle may start before the end of the map phase but finish only after all map tasks have finished. Sort and Reduce may only start when the shuffle completes. Since map and shuffle phases overlap, the coordination between them plays a significant role in determining the overall runtime. *mapreduce.job.reduce.slowstart.completedmaps* in mapred-site.xml is specified as a percentage of completed map tasks to control the time when reduce tasks start collecting the output of map tasks. The early shuffle incurs not only hanging reduce tasks, but also some delay in the execution of the map phase, because the map tasks and reduce tasks are competing for containers on the same physical machine. The general rule of thumb is that raising the default value (0.05) to be above 0.80 (80%) can increase resource utilization and decrease the reduce tasks waiting time (average shuffle time).

mapreduce.reduce.shuffle.parallelcopies is another network relevant parameter, which determines the number of parallel threads run by each reduce task during the shuffle phase. There is a tradeoff in setting this property: a low value will waste network bandwidth and CPU cores; a high value can cause not only too many pending threads but also network congestion. The near-optimal value is dependent on the network bandwidth as well as the number of CPU cores on each node.

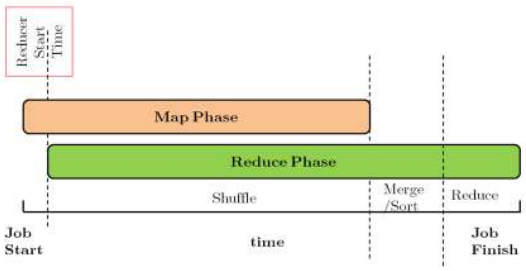


Figure 2: Decomposition of Reducer Phases

4. Initial Evaluation

Hadoop provides a default value for each parameter. However, the default configuration is normally insufficient for most Hadoop workloads and degrades the Hadoop job performance. For example, our data node has 32 CPU cores and 60GB memory, but the maximum number of CPU cores and the maximum memory that can be allocated for containers on each node are only 8 and 8GB respectively by default. As a result, the remaining 24 CPU cores and 52GB memory will be idle because no work load will be assigned to them. For this reason, we will not tune parameters based on the default settings because it is too low. Our first step is to pinpoint a initial tuned level as a reference for our next fine-tuning part. The goal is to keep the resource utilization of cluster balanced and get an acceptable performance given our hardware resource (e.g. CPU, memory, disk, network).

4.1. Experimental Setup and Workloads

To evaluate k-mer counting performance, we use a cluster of 15 EC2 c3.8x large machines (1 master, 14 workers) each with 32 cores, 60 GB memory and an 500 GB SSD. All the machines are connected by 10Gbit/s Ethernet. We choose cow rumen metagenomics dataset [4], the same dataset used in the original BioPig paper [20], as our test input. From the I250 serial, the one with the largest size and the best quality, we generate 6 different workloads: 1GB, 10GB, 20GB, 40GB, 60GB and 100GB. The k-mer size is fixed to 20 for simplicity as in the original BioPig paper [20].

4.2. Initial Tuned Configuration and Performance

With Pig, users must specify the number of reduce tasks. After running a bunch of jobs with various number of reducers, estimations of reducer numbers at various workloads are found and shown in Table 3.

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

Table 3: Numbers of Mappers and Reducers for Different Data Size

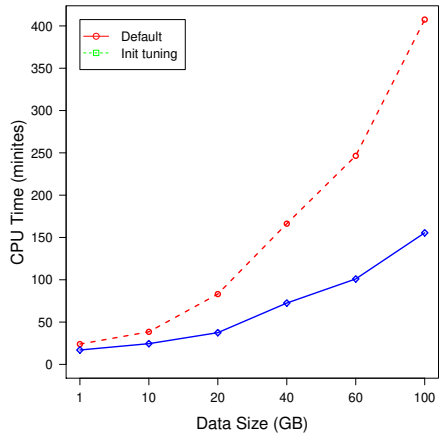


Figure 3: Default and Baseline Performance

Based on the general tuning guidelines provided in section 3, values of those significant Hadoop parameters are set as in Table 4. For succinctness, we avoid redundant information provided in the previous section. On our EC2 cluster, we use the default block size (64MB) and allocate 28 containers on each node. More precisely, for each container, we allocate 2GB memory and 1624MB JVM heap size. 1000MB is allocated for sorting the map output within the JVM heap size. Reduce tasks start when 80% of map tasks have finished.

Using the initial tuned configuration and the above set number of reducers, k-mer counting is executed with 6 different data sizes. The results (figure 3) show that with the initial tuned parameters, the job execution time is decreased by about 60% compared to the default settings. Our analysis of the Hadoop counters/logs reveals that the initial tuning configuration doesn't fully utilized the hardware resource. In the next section, we will fine-tune those significant parameters based on the initial configuration.

5. Fine-tuning

Our tuning follows an iterative process and focuses on the significant parameters (in section 3). In an effort to provide a MapReduce framework that can be used by users without an in-depth understanding of Hadoop internals, Hadoop provides a rich web interface for managing Hadoop clusters and MapReduce jobs. The ResourceManager (default port 8088) and JobTracker (default port 19888) web interfaces are two most used ones for diagnosing and troubleshooting performance bottlenecks, since they provide a wealth of information (such as counters, logs, etc.) on jobs and tasks that are running on the cluster as well as historical information on completed jobs. In each iteration, we run the job, identify a bottleneck from these web interfaces and then adjust relevant parameters. This tuning cycle starts off at our previous initial tuning configuration and repeats until all the observed problems are addressed. We mainly go over the following 4 steps: compressing the output of the map and reduce tasks, reducing the cost of data spilling, minimizing the overhead of garbage collection, and coordinating of the map and reduce phases. The detail of these tuning steps is discussed in section 2.

5.1. K-mer Counting Characteristics

One major overhead for data-intensive applications is the intermediate data. The pecularity of k-mer counting lies in its exceptionally large intermediate data size relative to input data size. Tables 5 and 6 compare the ratio of intermediate data size to input data size between common Hadoop applications and k-mer counting. From these results, it can be perceived that k-mer counting application often generates more than ten-fold intermediate data relative to the input data. This distinct feature makes some of available Hadoop tuning guidelines inapplicable for k-mer counting. In our tuning effort, we use 40GB and 60GB input data sizes to fully utilize the cluster resource while allowing us to complete experiments within a reasonable amount of time and cost.

Job Name	Input data 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 5: 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: Characteristics of Intermediate Data for k-mer (k=20) Counting

5.2. Parameters Tuning

Compressing the Output of the Map and Reduce Tasks (Disk Relevant Parameter Tuning)

K-mer counting is considered I/O bound as it generates large amount of intermediate data. For such I/O bound

Configuration parameters	Default Value	Initial-tuned 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.maximum-allocation-vcores	32	30
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
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	0.80

Table 4: Initial-tuned Configuration

applications, data compression and decompression trade off CPU cycles for reduced I/O costs. The smaller intermediate data size not only reduces the number of local disk operations each map and reduce task perform but also reduces network transfers from map tasks to reduce tasks.

Hadoop supports multiple compression formats (zlib, gzip, LZO, bzip2, Snappy, etc). Because of the good balance between speed and space, Snappy was chosen for this test. As can be seen from Table 7, data compression yielded more than 50% decrease in disk I/O. For 40GB workload, the number of bytes read and written was decreased from 604GB to 283GB and from 1200GB to 550GB, respectively. Consequently, the overall job time was decreased by 6 minutes for 40GB input and 8 minutes for 60GB input. Our testbed (EC2 cluster) is equipped with SSDs. Thus the performance gain of HDD is more significant compared to clusters with SSD as the primary storage.

Reducing the Cost of Data Spilling (Disk Relevant Parameter Tuning)

In Hadoop, data is split into blocks. These blocks are stored on the DataNodes of the HDFS file system. One map task is created for each block by default. Therefore, the number of map tasks is determined by block size and input data size. The larger the block size is, the fewer map tasks will be spawned in the Hadoop cluster. A large block size is supposed to be beneficial according to some studies [23, 26] because the fewer number of map tasks incurs lower overhead of starting up and tearing down. However, k-mer counting is an exception due to the high merging overhead brought by its large intermediate data size. Our analysis of the spill logs reveals that, with 64MB block size, each map task generated 3 spills. The overhead of merging those spilled records dwarfed the benefits of having large block sizes. We tried different block size (Figure 4) and found that the block size of 32MB yielded best performance. The overall job execution time was decreased by 12 minutes for 40GB input and 17 minutes for 60 GB input after reducing the block size to 32MB. However, the Hadoop counters shows that the spilled records in the map phase was still greater than the map output records, indicating more than one spills were generated by each map task. To avoid the penalty of merging the spilled records, we increased the map-side buffer size to 1100MB in such a way that all the intermediate data can be contained in the buffer. Spill mechanism implies that the performance is best in this scenario. Experimental results showed that more than 50% of I/O overhead was reduced by this tuning. Specifically, the total number of bytes read and written was decreased by 567GB and 886GB for 40GB and 60GB inputs respectively.

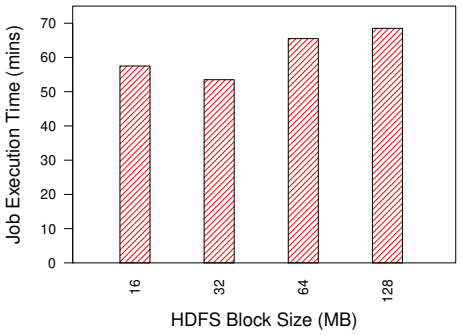


Figure 4: Effect of the HDFS Block Size

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

Table 7: IO Improvement from Data Compression

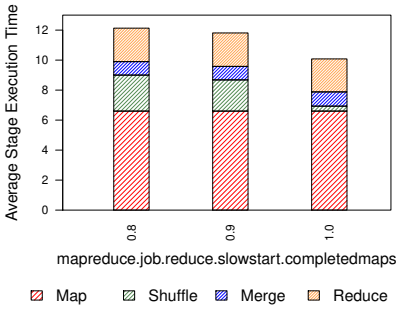


Figure 5: Impact of Reducer Start Time

Minimizing the Overhead of Garbage Collection (Memory Relevant Parameter Tuning)

In Hadoop, a JVM daemon is launched for each task. Java Garbage Collection (GC) is an automatic mechanism to manage the runtime memory by JVM. It is done by copying the survival objects from *Young Generation* [35] to *Permanent Generation* [35] when the former is full. Once the *Permanent Generation* is also filled up, the whole JVM heap is reclaimed during which all survival objects in the heap are collected, a process referred to as *full GC*. Full GC affects performance of Java applications and is a sign that some adjustments need to be made to the GC settings. Avoiding or minimizing full GC can prevent performance degradation caused by the memory swap. Our log showed that there was one *full GC* every 45 seconds, which, if not adjusted properly, would incur significant overhead to the application. By configuring the size of *Permanent Generation* from 20MB (default value) to 128MB, and the number of parallel threads for garbage collection to 4, the overall job run time was decreased by 9 minutes for 40GB input and 11 minutes for 60GB input as displayed in table 8.

Coordinating of the Map and Reduce Phases (Network Relevant Parameter Tuning)

From section 3, we know that we can configure the time until reduce tasks begin shuffling. Let us denote the percentage of completed map tasks when shuffling starts as the parameter value λ, which is controlled by *mapreduce.job.reduce.slowstart.completedmaps* parameter. The average shuffle time is a key indicator of whether the start time of the reduce tasks need to be reconfigured. With the initial setting ($\lambda = 0.80$), we analyzed the logs and found that the average shuffle time was still relatively long, which indicates that the performance suffered since the early-launched reduce tasks occupied the available resource (e.g. cores, memory associated with each container). After trying several different values (Figure 5), we set λ to 1.00 for the k-mer counting application, meaning that the reduce tasks will only start when all the map tasks are complete. Result shows the overall job execution time was decreased by 4 minutes for 40GB input and 8 minutes for 60GB input by this tuning. Note that our EC2 nodes are connected by 10Gigabit Ethernet, which reduces the bandwidth bottleneck. In this case the slowstart parameter can be set very large (0.999 or 1.0). For the Hadoop clusters with lower-bandwidth interconnect, e.g. 1Gb Ethernet, such a high value can cause sudden spikes in network traffic and create network slowdown, since all the map tasks start copying at the same time.

5.3. Discussions

Figure 6 summerizes how each tuning step affects the overall runtime in detail. Among them, spilled records reduction and full GC elimination are the two most significant steps. In addition, the performance improvement by tuning these factors is roughly proportional to the data size.

The performance difference before and after fine-tuning can be seen in figure 7. More precisely, with the fine-tuned parameters, the overall runtime is reduced by 44% for the 40GB input, and 47% for the 60GB input compared to the initial configuration (in section 4). In connection with the default settings, we have achieved a total of 4x speedup. Aside from these performance improvement, the linear scalability is also kept during the tuning process.

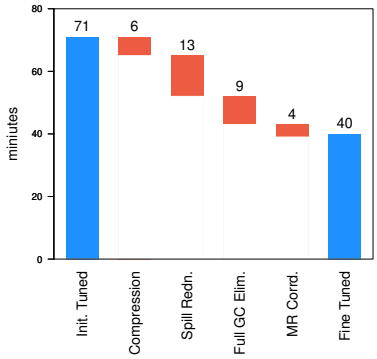
On the other hand, disk I/O and network bandwidth are usually two performance bottlenecks for Hadoop applications. SSD and 10Gigabit Ethernet for our EC2 cluster help mitigate the impact of these constraints. We believe that applying these tunings to Hadoop clusters with slower network and slower storage may bring more performance improvements.

6. Related Work

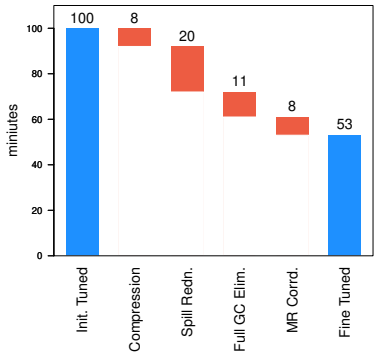
Hadoop MapReduce tuning has been studied in [23, 24, 25, 26, 28, 29]. [23, 24] and [26] provide general guidelines to tune parameters on Hadoop 1. In contrast, we not only provide some valid suggestions on the next generation of the Hadoop computing platform known as YARN, but also present a detailed step-by-step practical

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 8: Performance Gain from GC Tuning



40GB



60GB

Figure 6: Impact Factors

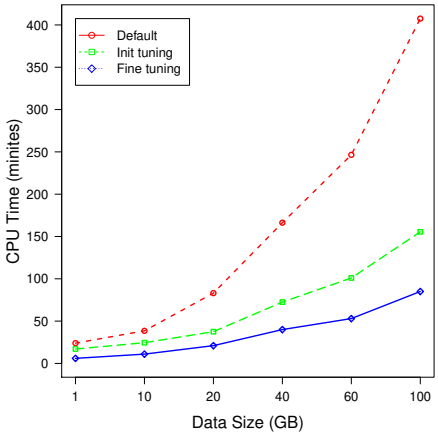


Figure 7: Performance Comparison

experience for tuning applications in the domain of bioinformatics. In addition, [23, 24] and [26] all focus on the built-in benchmarks of the Hadoop framework, among them TeraSort is one of the most extensively studied workloads. Bioinformatics applications have quite unique characteristicsas (as shown in section 5), which makes some of the available tuning guidelines inapplicable. For instance, both [23] and [26] claim that a larger block size can bring better performance because of the lower overhead of map task creation and destruction . However, for our k-mer counting tuning, big block size causes the map tasks spill heavily to disk which seriously impairs the performance.

[25, 28] and [29] present auto-tune sytems which are designed to tune Hadoop parameters automatically. Although these solutions may at first appear to be viable and attractive, the core problem is that they often lack efficiency and effectiveness to capture the characteristics of all application domains. Furthermore, the large parameter space, with its complex trade-offs and inter-dependencies, and multi-tenant Hadoop cluster environments increase the complexity of auto tuning, in which not only some models are used to estimate job execution time but also some search algorithms are designed to find the optimal settings. Different from these studies, to simply the tuning problem, we reduce the parameter dimensionality and classify the significant parameters in this work. In practice, users can easily identify major bottlenecks by checking Hadoop counters, thereafter our approach limits the search to the significant parameters and help them quickly

identify the best parameters settings. The near-optimal performance can be achieved by following the iterative process of addressing bottlenecks. Compared to the rigid, predefined auto-tune systems, our methodology is not only adaptive and flexible to any particular application on a given cluster, but also can deliver better performance.

Asides from the dimension reduction of parameter space and significant parameters classification, this is the first study, to our knowledge, to emphasize the importance of the application characteristics and present a practical reference for tuning applications in the bioinformatics domain. In this regard, our paper not only offers a valuable case study, but also brings a new perspective on Hadoop tuning.

7. Conclusion and Future Work

With the growing popularity of cloud computing and exponential data growth in bioinformatics, an increasing number of Hadoop MapReduce applications are written for large-scale, data-intensive analysis of bioinformatics datasets. Hadoop configuration optimization is a challenging problem that requires user experience and deep knowledge of the Hadoop framework. In this paper, to reduce the complex scope of Hadoop parameter tuning and provide biological scientists a clear direction, we ignore the parameters that have litter performance effect and classify the significant parameters into 4 groups. We use k-mer counting as our study case and focus on tuning its significant parameters. According to k-mer counting characteristics, 4 steps are conducted to further improve our EC2 cluster utilization based on an initial tuning configuration. Results show that these tunings achieve a total of 4x performance improvement compared to the default settings.

Even though parameter tuning can bring obvious performance improvement, I/O is still a major bottleneck of Hadoop-based applications. Future experiments to reduce I/O may include implementing a combiner to reduce the amount of transferred data to the reducers, reimplementing k-mer counting analysis on Apache Spark [36], or using Apache Tez [37] as Pig's execution engine.

Acknowledgments

We are very thankful to Dr. Shane Canon from Lawrence Berkeley National Lab, Mr. Brandon Stephens from Florida State University, and the anonymous reviewers for their insightful comments. This work is funded in part by National Science Foundation awards 1561041 and 1564647. Xiandong Meng, Zhong Wang, and Lizhen Shi partially, are supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] J. Zhang, R. Chiodini, A. Badr, G. Zhang, The impact of next-generation sequencing on genomics, Journal of genetics and genomics 38 (3) (2011) 95–109.
[2] Encode Project, https://www.encodeproject.org/.
[3] 1000 Genomes Project, http://www.1000genomes.org/.
[4] M. Hess, A. Sczyrba, R. Egan, T.-W. Kim, H. Chokhawala, G. Schroth, S. Luo, D. S. Clark, F. Chen, T. Zhang, et al., Metagenomic discovery of biomass-degrading genes and genomes from cow rumen, Science 331 (6016) (2011) 463–467.
[5] Human Microbiome Project, https://en.wikipedia.org/wiki/Human_Microbiome_Project/.
[6] Hadoop, https://hadoop.apache.org/.
[7] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.
[8] AWS, http://aws.amazon.com/.
[9] A. Matsunaga, M. Tsugawa, J. Fortes, Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications, in: eScience, 2008. eScience'08. IEEE Fourth International Conference on, IEEE, 2008, pp. 222–229.
[10] M. C. Schatz, Cloudburst: highly sensitive read mapping with mapreduce, Bioinformatics 25 (11) (2009) 1363–1369.
[11] S. Leo, F. Santoni, G. Zanetti, Biodoop: bioinformatics on hadoop, in: Parallel Processing Workshops, 2009. ICPPW'09. International Conference on, IEEE, 2009, pp. 415–422.
[12] B. Langmead, M. C. Schatz, J. Lin, M. Pop, S. L. Salzberg, Searching for snps with cloud computing, Genome Biol 10 (11) (2009) R134.
[13] 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) (2010) 1297–1303.
[14] B. Langmead, K. D. Hansen, J. T. Leek, et al., Cloud-scale rna-sequencing differential expression analysis with myrna, Genome Biol 11 (8) (2010) R83.
[15] 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) (2010) R86.
[16] L. Pireddu, S. Leo, G. Zanetti, Seal: a distributed short read mapping and duplicate removal tool, Bioinformatics 27 (15) (2011) 2159–2160.
[17] T. Nguyen, W. Shi, D. Ruden, Cloudaligner: A fast and full-featured mapreduce based tool for sequence mapping, BMC research notes 4 (1) (2011) 171.
[18] Contrail, http://www.homolog.us/blogs/blog/2011/09/08/contrail-a-de-bruijn-genome-assembler-that-uses-hadoop/.
[19] 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) (2012) 721–723.
[20] H. Nordberg, K. Bhatia, K. Wang, Z. Wang, Biopig: a hadoop-based analytic toolkit for large-scale sequence data, Bioinformatics (2013) btt528.
[21] A. Schumacher, L. Pireddu, M. Niemenmaa, A. Kallio, E. Korpelainen, G. Zanetti, K. Heljanko, Seqpig: simple and scalable scripting for large sequencing data sets in hadoop, Bioinformatics 30 (1) (2014) 119–120.
[22] D. Decap, J. Reumers, C. Herzeel, P. Costanza, J. Fostier, Halvade: scalable sequence analysis with mapreduce, Bioinformatics (2015) btv179.
[23] D. Heger, Hadoop performance tuning-a pragmatic & iterative approach, CMG Journal 4 (2013) 97–113.
[24] S. B. Joshi, Apache hadoop performance-tuning methodologies and best practices, in: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ACM, 2012, pp. 241–242.
[25] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, N. Fuller, Mronline: Mapreduce online performance tuning, in: Proceed-

ings of the 23rd international symposium on High-performance parallel and distributed computing, ACM, 2014, pp. 165–176.
[26] P. TUNING, Performance tuning.
[27] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A self-tuning system for big data analytics., in: CIDR, Vol. 11, 2011, pp. 261–272.
[28] G. Liao, K. Datta, T. L. Willke, Gunther: Search-based autotuning of mapreduce, in: Euro-Par 2013 Parallel Processing, Springer, 2013, pp. 406–419.
[29] P. Lama, X. Zhou, Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud, in: Proceedings of the 9th international conference on Autonomic computing, ACM, 2012, pp. 63–72.
[30] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache hadoop yarn: Yet another resource negotiator, in: Proceedings of the 4th annual Symposium on Cloud Computing, ACM, 2013, p. 5.
[31] W. Yu, K. J. Wu, W. Ku, C. Xu, J. Gao, BMF: bitmapped mass fingerprinting for fast protein identification, in: 2011 IEEE International Conference on Cluster Computing (CLUSTER), Austin, TX, USA, September 26-30, 2011, 2011, pp. 17–25.
[32] Sequence Analysis, `https://en.wikipedia.org/wiki/Sequence_analysis`.
[33] C. Olston, B. Reed, U. Srivastava, R. Kumar, 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.
[34] de Bruijn graphs, `https://en.wikipedia.org/wiki/De_Bruijn_graph`.
[35] V. L. Shrinivas Joshi, Java garbage collection characteristics and tuning guidelines for apache hadoop terasort workload.
[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Vol. 10, 2010, p. 10.
[37] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, C. Curino, Apache tez: A unifying framework for modeling and building data processing applications, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, 2015, pp. 1357–1369.