Performance Evaluation and Tuning of BioPig for Genomic Analysis

Lizhen Shi
Florida State Univ.
lshi@cs.fsu.edu

Zhong Wang
Joint Genome Inst.
zhongwang@lbl.gov

Weikuan Yu
Florida State Univ.
yuw@cs.fsu.edu

Xiandong Meng
Joint Genome Inst.
xiandongmeng@lbl.gov

ABSTRACT

In this study, we aim to optimize Hadoop parameters to improve the performance of BioPig on Amazon Web Service (AWS). BioPig is a toolkit for large-scale sequencing data analysis and is built on Hadoop and Pig that enables easy parallel programming and scaling to datasets of terabyte sizes. AWS is the most popular cloud-computing platform offered by Amazon. When running BioPig jobs on AWS, the default configuration parameters may lead to high computational costs. We select the k-mer counting as it is used in a large number of next generation sequence (NGS) data analysis tools. We tuned Hadoop parameters from five different perspectives based on a baseline configuration. We found tuning different Hadoop parameters led to various performance improvements. The overall job execution time of k-mer counting on BioPig was reduced by 50% using an optimized set of parameters. This paper documents our tuning experiments as a valuable reference for future Hadoop-based analytics applications on genomics datasets.

1. INTRODUCTION

Next-generation sequencing technologies [12] have increased the speed of DNA sequencing by several orders of magnitude. Automation and computerization [3] revolutionized the speed of reading the letters of DNA sequences. The advent of next-generation sequencing technique increased it further. In current days, modern Illumina [18] systems can generate hundreds of gigabytes of sequences per run with 99.9% accuracy. This gives rise to a large amount of raw sequence datasets that need to be processed, hence placing a huge burden on external compute framework.

Meanwhile, extremely large scale sequencing projects are emerging, such as ENCODE project [4], 1000 Genomes project [1], Cow Rumen Deep Metagenomes project [9] and Human Microbiome project [5]. These projects produce sequencing data at a massive scale. Traditional analysis tools have been challenged by the need to scale commensurately with the increasing data size.

Hadoop [2] has emerged as a popular option to address the challenges of increasingly large datasets. The 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, scalable and fault tolerance data storage, but also built-in data locality and distributed data processing. MapReduce permits tasks running 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. The advantages of Hadoop over other parallel frameworks make it widely adopted by various scientific domains including bioinformatics. Biological scientists harness the power of Hadoop and large clusters to analyze large sequence data sets.

However, one issue with efficiently leveraging MapReduce is its large number of parameter configurations. Tuning all parameters is typically very time-consuming. Although there has been a lot of research on this topic [8, 10, 11, 16], none has provided a solid guidance for the analytics applications to achieve an optimal performance. This is because the interplay of parameters is typically dependent on the input data size, hardware resource and application characteristics. Even worse, available MapReduce tuning guidelines are not applicable to application of bioinformatics due to its own unique characteristics. We use BioPig [13], one of the Hadoop-based toolkits for next-generate sequence (NGS) analysis in bioinformatics, as a representative application for tuning Hadoop parameters. Based on BioPig application characteristics, our tuning was conducated from five perspectives: data compression, block size, heap size, JVM garbage collection and reducer start-time. With well-tuned parameters, the overall job execution time was decreased by 50% compared to baseline configuration. We believe that this tuning experience can provide a valuable reference for other similar applications that analyze large-amounts of NGS datasets in the cloud.

The paper is organized as follows. Section 2 introduces the background of Hadoop and BioPig. Section 3 gives an initial evaluation of BioPig, followed by its tuning in Section 4. Section 5 provides a review of related work. Section 6 concludes the paper and discusses some future directions to optimize BioPig performance further.

2. BACKGROUND

In this section, we first review the two core components

DISCS-2015, November 15-20, 2015, Austin, TX, USA
 ISBN 978-1-4503-3993-3/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2831244.2831252

in Hadoop: Hadoop Distributed File System (HDFS) and MapReduce programming model. Next we introduce DNA sequence analysis and BioPig toolkit.

2.1 Overview of the Hadoop Framework

Apache Hadoop as an open source implementation of 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)

Using commodity components, Hadoop can process data in a distributed Hadoop cluster. Data are broken into blocks and distributively stored at the data nodes. Hardware failure may occur on large-scale systems. To achieve high availability, each block of data is replicated to 3 data nodes in Hadoop to prevent the failure of one node from causing the loss of valuable data. To improve data processing performance, HDFS supports data locality, which means the workload would be assigned to the node where the data is stored. This helps decrease network overhead and increase the scalability of Hadoop-based data analytics.

HDFS includes two types of nodes: a NameNode and many DataNodes. Even though the NameNode does not hold any dataset, it holds the metadata for the Hadoop cluster and monitors the health of the DataNodes through the heartbeats from DataNodes. As a result the NameNode is the most critical component of HDFS. Prior to Version 2.0, Hadoop allows only a single NameNode for the entire cluster and the cluster size is limited because of the associated bottleneck. In Version 2.0, federation is employed to address this limitation, where multiple NameNodes are allowed.

2.1.2 Hadoop MapReduce

MapReduce [7] provides a programming model for data-intensive processing in a 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. Users may use a general-purpose programming language such as Java or Python to implement customized Map and Reduce functions for the processing logics needed by their own applications.

Hadoop MapReduce is in its second generation, which we call MapReduce 2 (MRv2) or YARN. YARN was developed to overcome several limitations in the first generation of Hadoop. First, instead of dividing resources into map and reduce slots, it provides a unified resource unit called containers. This allows flexible scheduling. Second, it relieves the single bottlenecked JobTracker through a hierarchical management scheme in which the resource manager (RM) is responsible for the global coordination within the system and an ApplicationMaster (AM) per job is created per job and manages all tasks with the job. Finally, YARN supports both MapReduce and non-MapReduce applications as a new resource management framework. Taken together, compared to the original Hadoop, YARN provides flexible resource management and versatile programming models.

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 [17, 6].

BioPig [13] 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 [14] data flow language. Leveraging the advantages of Hadoop and Pig framework, BioPig has shown its scalability, programmability and portability. Through this work, 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. Among these modules, K-mer counting is the core one, which is also 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 analysis, such as constructing de Bruijn graphs 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 the scalability issue, BioPig framework provides a scalable k-mer counter which scales well with the dataset size and k-mer size because of the linear scalability of Hadoop framework.

3. AN INITIAL EVALUATION OF HADOOP-BASED BIOPIG

Hadoop has been adopted widely by various scientific domains. However, one challenge with Hadoop is that its performance tuning is very time-consuming. Hadoop programs have a very large set of configuration parameters. These parameters are used to control the execution behavior of jobs. Many of them can affect the job's execution time and resource usage significantly.

In this section, we will take k-mer counting as an example to describe the evaluation of Hadoop-based BioPig and demonstrate its performance characteristics. The evaluation provides a baseline for the tuning of BioPig that will be described in Section 4. We use the default value for the configuration parameters unless a choice is explicitly specified.

3.1 System Configuration

Our initial evaluation is to pinpoint the baseline of configuration parameters for a reference about BioPig performance. The goal is to get an acceptable performance providing the limitations of hardware resources (e.g. CPU, memory, disk, network). We first describe the hardware and software configurations on our system, and the input data preparation,

To evaluate BioPig performance, 15 nodes (1 name node and 14 data nodes) of the type c3.8x large instances were launched on Amazon 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

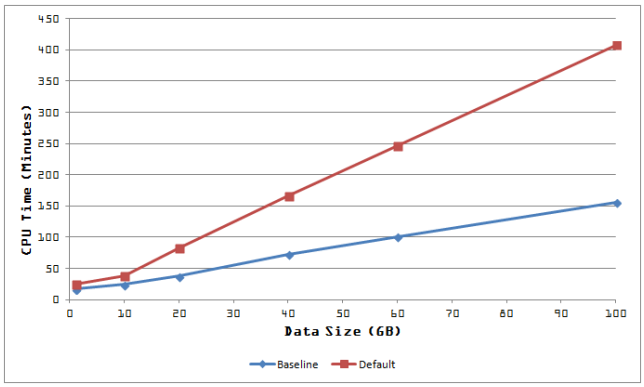

Figure 1: Default and Baseline Performance

attached to each node as local and HDFS storage for MapReduce jobs. All the nodes are interconnected by 10Gbit/s Ethernet network. We choose cow rumen metagenomics dataset [9], the same dataset used in the original BioPig paper [13], as our test input. From the I250 serial, the one with the largest size and the best quality fasta file, we generated 6 different workloads: 1GB, 10GB, 20GB, 40GB, 60GB and 100GB. The k-mer size was fixed to 20 for simplicity.

Based on the tuning of other Hadoop applications, values of some critical Hadoop parameters are set as shown in Table 1. These parameters are stored in these three files: hdfs-site.xml, mapred-site.xml and yarn-site.xml. The differences between baseline settings and default values are also shown in the table.

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

- We set 2GB memory as the limit for Map containers and Reduce containers.
- We allocate a max Java heap size of 1624MB for Map/Reduce processes.
- A buffer size of 1000 MB is allocated for storing outputs for MapTasks.
- We allocate 28 containers on each node with a default block size of 64MB.

3.2 Initial Performance Evaluation

Each node in the AWS cluster has 60 GB of RAM. YARN can only allocate up to 58,296 MB of memory to all containers. The remaining memory is reserved for the OS kernel, system processes and other non-Hadoop processes. After running a bunch of jobs with a various number of reducers, estimations of reducer numbers at various workloads are found and shown in Table 2.

Using the baseline configuration and this set number of reducers, k-mer counting was executed with 6 different data sizes. The results are shown in Figure 1. The performance difference between the default and baseline configuration is also shown in the figure. Then the counters and logs of these jobs were collected and analyzed. Based on the performance results and our examination, the hardware resources were not fully utilized by the basline configuration. In the next section, we describe the tuning of BioPig from five perspectives.

4. TUNING BIOPIG

Our tuning process follows an iterative approach. In each iteration, we run the job, identify a bottleneck from Hadoop counters/logs and then adjust parameters. This process starts with the baseline configuration and repeats until all major bottlenecks are addressed . Data compression, block size, heap size, JVM garbage collection and reducer start-time are the dominating factors that affect the performance. Our work centers on tuning these steps of k-mer counting discussed in Section 2.

4.1 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 3 and 4 compare the ratio of intermediate data size to input data size between common Hadoop applications and k-mer counting application. 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 most of available Hadoop tuning guidelines inapplicable for k-mer counting. In our tuning effort, we use 40GB and 60GB input data size to fully utilize the cluster resource while allowing us to complete experiments within a reasonable amount of time.

4.2 Parameters Tuning

4.2.1 Data Compression

For data-intensive 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 accesses of each MapTask and ReduceTask but also network transfers from MapTasks to ReduceTasks.

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 5, data compression yielded more than 50% decrease in disk IO. For 40GB workload, the read and write amount 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 SSD. Thus the performance gain of HDD is more significant compared to clusters with SSD as the primary storage.

4.2.2 Block Size and Heap Size

In Hadoop, data is split into blocks. These blocks are stored on the data nodes of HDFS file system. One MapTask is created for each block by default. Therefore, the number of MapTasks is determined by block size and input data size. The larger the block size is, the fewer MapTasks will be spawned in the Hadoop cluster. A large block size is supposed to be beneficial according to some studies [8, 16] because the fewer number of MapTasks 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. To be specific, a MapTask 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 back-

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 1: Baseline Configuration

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 2: Numbers of Mappers and Reducers for Different Data Size

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 3: 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 4: Characteristics of Intermediate Data for k-mer Counting (k=20)

ground thread. One MapTask may generate multiple spills depending on the buffer size and Map output data size. If more than one spills are generated, the spilled data have to be merged into a single sorted file partitioned by Reduce keys. Then ReduceTasks pull their input from this partitioned merged file via network. The details of this process are shown in Figure 2. Analysis of logs demonstrated that large block size led to more map-side spills due to the limited buffer size of each MapTask. The overhead of merging these spills dwarfed the benefits of small block size. Our experiment demonstrated that block size of 32MB yielded better performance than block size of 64MB. The overall job execution time was decreased by 12 minutes for 40GB input and 17 minutes for 60 GB input after changing the block size from 64MB to 32MB. spill mechanism also implies that the performance is best when the intermediate data is well contained in the sort buffer. Our analysis shows that increasing map-side heap to 1100MB can help eliminate spill operations in our experiments. Table 6 shows that more than 50% of IO overhead was reduced by this tuning.

4.2.3 JVM Garbage Collection

In Hadoop, a JVM daemon is launched for each task. Java

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 5: IO Improvement from Data Compression

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: IO Reduction from Map-side Spill Tuning

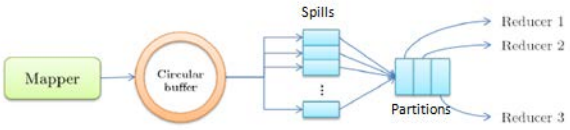

Figure 2: Shuffle and Sort Process

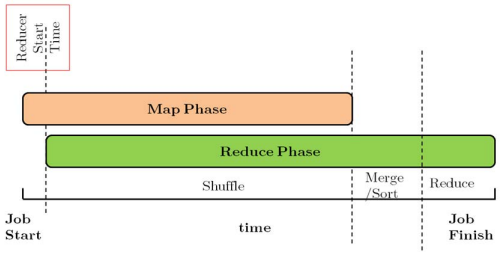

Figure 3: Decomposition of Reducer Phases

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* [15] to *Permernant Generation* [15] when the former is full. Once the *Permernant 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*. 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 *Permernant 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 7.

4.2.4 Reducer Start Time

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 MapTasks 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 all the MapTasks, incurs heavy network traffic because each ReduceTask pulls its input from almost every other nodes and writes its output into HDFS. More precisely the Reduce phase can be divided into 3 steps:

Shuffle: Collects input from MapTasks.

Sort: Sorts the records and merges them by keys.

Reduce: Runs the Reduce program, then writes its result into HDFS.

Figure 3 shows the timeline of a MapReduce job execution. Shuffle may start before the end of the Map phase but only finish after all MapTasks have finished. Sort and Reduce may only start when the shuffle completes. Since map and shuffle phases overlap, the coordination of them is crucial for the overall job execution time. The time that a ReduceTask begins shuffling is configurable. Let us denote the percent of completed MapTasks when shuffling starts as the parameter value λ, which is controlled by *mapred.reduce.slowstart.completed.maps* parameter (default 5%) in mapred-site.xml. When λ is low, the performance can suffer since the early-launched ReduceTasks can waste the available resource (e.g. cores, memory associated with to each container). After having tried a set of different values (Figure 4), we set λ to 1.00 for the k-mer counting application. The overall job execution time was decreased by 4 minutes for 40GB input and 8 minutes for 60GB input. Note that our EC2 nodes are connected by 10Gigabit Ethernet, which reduces the bandwidth bottleneck. For the Hadoop clusters with lower-bandwidth interconnect, e.g. 1Gb Ethernet, we believe λ can be selected following a similar tuning approach.

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 7: Performance Gain from GC Tuning

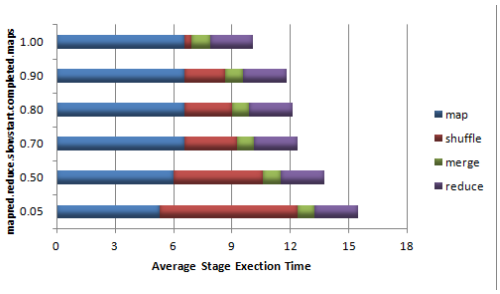

Figure 4: Impact of Reducer start time

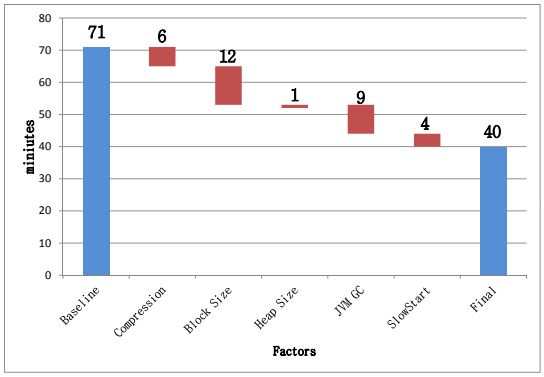

Figure 5: Impact Factors on 40GB

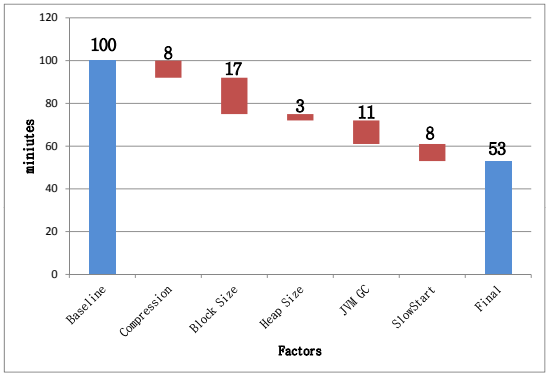

Figure 6: Impact Factors on 60GB

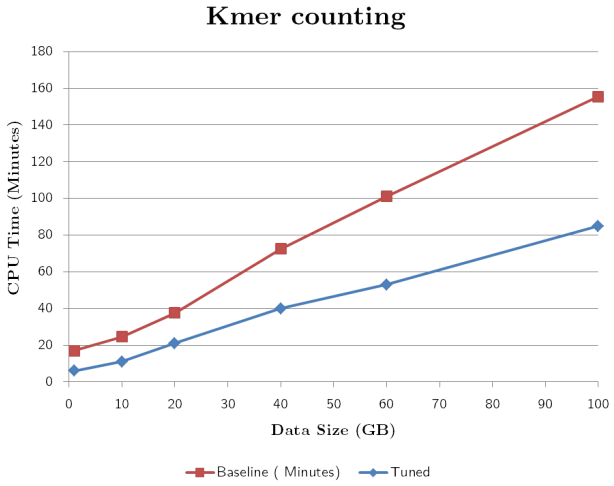

Figure 7: Performance Comparison

4.3 Discussions

Figure 5 and Figure 6 summerize how each tuning step affects the overall run time in detail. Block size and JVM garbage collection are two most significant factors and the performance improvement by tuning these factors is roughly proportional to the data size.

Figure 7 shows the performance trend before and after tuning. The overall job execution time is reduced by 44% for the 40GB input and 47% for the 60GB input. The linear scalability is kept.

On the other hand, disk IO 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 slow networks may bring more performance improvements.

5. RELATED WORK

Hadoop tuning has been studied in [8, 10, 11, 16]. [16] provided general suggestions to tune parameters on Hadoop 1. However the suggestions are too general to fit specific applications. Heger et al. [8] and Joshi et al. [10] tuned TeraSort application on Hadoop 1 from three perspectives of hardware configuration, OS configuration and Hadoop parameters. Joshi et al. [10] even tuned the BIOS settings. In contrast, we not only presented a detailed step-by-step tuning process on the current Hadoop version YARN, but also provided some valid suggestions based on the unique workload characteristics. Li et al. [11] presented an online performance tuning system called MRONLINE. However, it can only delieve up to 30% performance improvement compared to default configuration. In contrast, our baseline perfor-

mance was increased by 60% compared to default settings and the final tuned performance was improved about 50% compared to the baseline configuration. As a result we obtained a total of 4x speedup with these tuned parameters.

Most of the Hadoop tuning studies have focused on common applications, TeraSort is one of the most extensively studied workloads among them. Bioinformatics applications have quite unique characteristics from these workloads, which makes most of the available tuning guidelines inapplicable.

For instance, both [8] and [16] claimed that larger block size can bring better performance because of the lower overhead of MapTask creation and destruction . However, for our BioPig tuning, big block size causes the MapTasks spill heavily to disk which seriously impairs the performance. In this regard, our paper not only offers a valuable tuning practice, but also brings a new perspective on Hadoop tuning.

6. CONCLUSION AND FUTURE WORK

The emergence of massive datasets in Bioinformatics poses great challenges in sequence analysis. The Hadoop MapReduce framework, which was designed to get its parallelism from large collections of commodity hardware, is adopted to address these challenges. Currently, several Hadoop-based Bioinformatics tools are available on the market. To improve BioPig performance, we tuned Hadoop parameters from 5 perspectives according to k-mer counting characteristics. Results showed these tunings achieved an average performance improvement of about 50% compared to baseline configuration.

Even though parameter tuning can bring obvious performance improvement, IO operation is still the bottleneck of Hadoop-based application. Future experiments to reduce IO may include implementing a combiner to reduce the amount of transferred data to the reducers, or reimplement BioPig's functions on Apache Spark[3], which is an in-memory computing framework.

Acknowledgments

We are very thankful to Dr. Shane Canon 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.

7. REFERENCES

[1] 1000 Genomes Project. `http://www.1000genomes.org/`.

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

[3] DNA Sequencing. `https://en.wikipedia.org/wiki/DNA_sequencing`.

[4] Encode Project. `https://www.encodeproject.org/`.

[5] Human Microbiome Project. `https://en.wikipedia.org/wiki/Human_Microbiome_Project/`.

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

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

[8] D. Heger. Hadoop performance tuning-a pragmatic & iterative approach. *CMG Journal*, 4:97–113, 2013.

[9] 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):463–467, 2011.

[10] S. B. Joshi. Apache hadoop performance-tuning methodologies and best practices. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 241–242. ACM, 2012.

[11] 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.

[12] M. L. Metzker. Sequencing technologies: the next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

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

[14] 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.

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

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

[17] W. Yu, K. J. Wu, W. Ku, C. Xu, and 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*, pages 17–25, 2011.

[18] 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.