eJAICT
ISSN 2355 1771
JOURNAL OF APPLIED INFORMATION, COMMUNICATION AND TECHNOLOGY
Published by:
Research Center of Technology and Innovation
Academic Research and Community Services
Swiss German University

Hadoop Configuration Tuning for Performance Optimization

Christian, Kho I Eng, Heru Purnomo Ipung

Department of Information Technology, Swiss German University, Tangerang 15143, Indonesia

Article Information

Received: 19 December 2016
Accepted: 17 February 2017
Published: 25 April 2017

DOI: 10.33555/ejaict.v4i1.81

Corresponding Author:

Christian
Email: christian286@student.sgu.ac.id

ISSN 2355-1771

ABSTRACT

Configuration parameter tuning is an essential part of the implementation of Hadoop clusters. Each parameter in a configuration plays a role that impacts the overall performance of the cluster. Therefore, we need to learn the characteristics of said parameter and understand the impact in hardware utilization in order to achieve optimal configuration. In this paper, we conducted experiments that includes modifying configuration and performed benchmark to find out if there is any performance gain. TeraSort is the program that runs the benchmark, we measure the time needed to complete the sort of the set of data and the CPU utilization during the benchmark. We conclude that from our experiments we can see significant performance improvements by tuning with the configurations. However, the results may vary between different cluster configuration.

Keywords: *apache hadoop, computer cluster, configuration tuning, terasort benchmark***.**

1. Introduction

Nowadays it is a hurdle to process large amounts of data or render a video because it takes a lot of time due to its requirement in processing power. Here in SGU, we do not have any supercomputer to overcome this issue as it is cost ineffective and high-power demand. A cheap and efficient alternative to a supercomputer is to have a High-Performance Computing (HPC) cluster using Apache Hadoop framework. By using this framework, we can create a cluster of computers to distribute heavy workload. Unfortunately, a clean install of a Hadoop cluster cannot guarantee its full potential performance of processing power. This can be addressed by identifying problems within the cluster and then applying the optimization software or hardware-wise.

By doing this research, we hope to achieve the full potential of our existing cluster. It may also prove that optimizing the configuration of the cluster could lead to better or worse performance. However, the results may vary depending on the hardware configuration, the hardware provided for this research purposes are a set of 4 consumer grade desktop PC with different specifications.

The default Hadoop configuration is not suitable for any cluster configuration on any scale as mentioned in [1]. Therefore, proper configuration is a part of implementation of Hadoop Clusters. Furthermore, it is likely that newer or older versions of Hadoop has configuration parameters that are deprecated.

2. Experiment Setup

Our cluster consists of 4 nodes which consists of single master node and 4 slave nodes. A gigabit switch is dedicated to providing connectivity to the nodes. Each node has 4 cores and at least 4 GB of RAM.

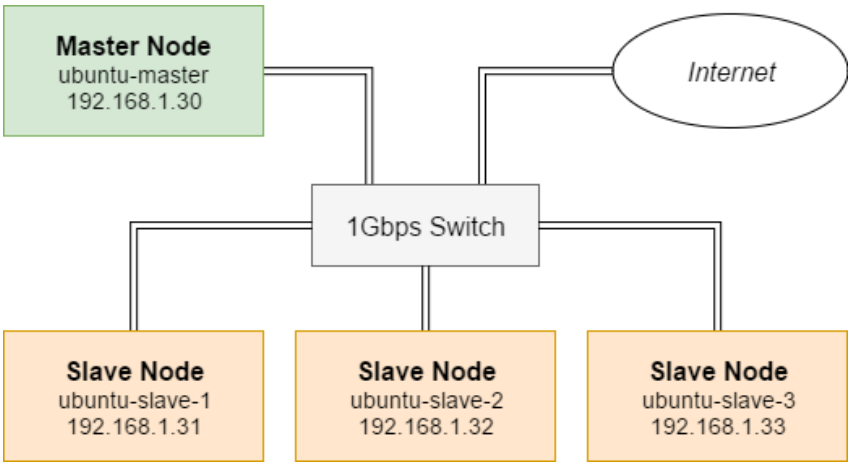


Figure. 1. *Network Map*

To create a baseline for our future experiments, we conducted an initial benchmark so that we compare the results with our further experiments. The initial benchmark is performed using bare minimal configuration to run the cluster. However, the recommended default configuration value is used if none is defined. The tools that are used in our benchmark are TeraGen, TeraSort, and TeraValidate. We provided 3 dataset that we generated using TeraGen with the size of 1 GB, 10 GB, and 50 GB that will be used throughout our experiments.

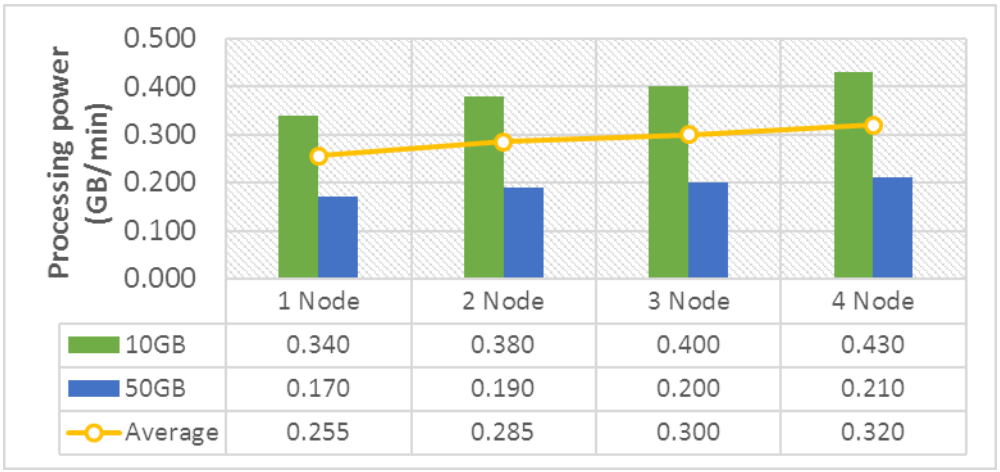


Figure. 2 Initial benchmark result

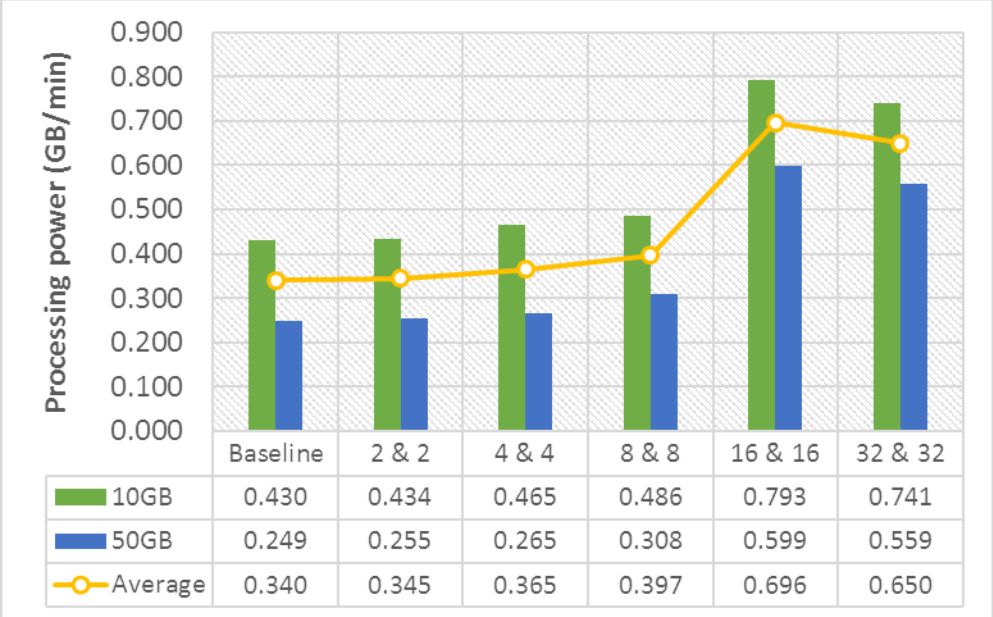
3. Experiments and Results

In this section, we present our results on the experiments that we conducted on our Hadoop cluster. The results are then compared with the baseline to determine if there is any performance improvement after tuning with the configuration. Our experiments includes changing the number of mapper and reduces, using native library instead of the provided java class, tuning the value of HDFS block size, and using different type of compression.

3.1. Results of Mapper and Reduces number tuning

Defining the number of mappers and reduces defines the number of task that can be carried out at the same time. Mappers accounts for the number of map job while reducers determine the number of reduce job. Since the number of mapper and reducer affects the utilization of the resources, it is essential to set the value to maximize the resource utilization. However, current system is limited to static allocation of resources meaning that the number of mappers and reduces can only be defined at the initialization of a job [2].

In this experiment, we need to determine the number of mapper and reduces that utilize the most resources. The value that we will use to benchmark is 2^n, n being the number of increments in our benchmark. The best value is determined when the next increment has little to no improvement compared to the previous increment.



	Baseline	2 & 2	4 & 4	8 & 8	16 & 16	32 & 32
10GB	0.430	0.434	0.465	0.486	0.793	0.741
50GB	0.249	0.255	0.265	0.308	0.599	0.559
Average	0.340	0.345	0.365	0.397	0.696	0.650

Figure. 3. Mapper and Reduces tuning result

From Figure. 3, we can see that as the number of mapper and reduces is increased, the time needed to process 10GB of data is faster. The steep incline being the jump from 8 reducers to 16 which cuts around 400 seconds of processing time. However, there is no improvement beyond 16 mappers and reduces.

3.2. Using native Hadoop Library

By default, the Hadoop library that is supplied with the binary package is compiled with the 32bit architecture of Linux. Since our operating system is using 64bit architecture, Hadoop must alternate using the built-in Java classes. This can harm the resource utilization as Java is more resource dependent. To resolve this issue, we either need to install the support of 32bit architecture library or to compile our own Hadoop source using 64-bit compiler. We are determined to recompile the Hadoop binary using 64-bit compiler.

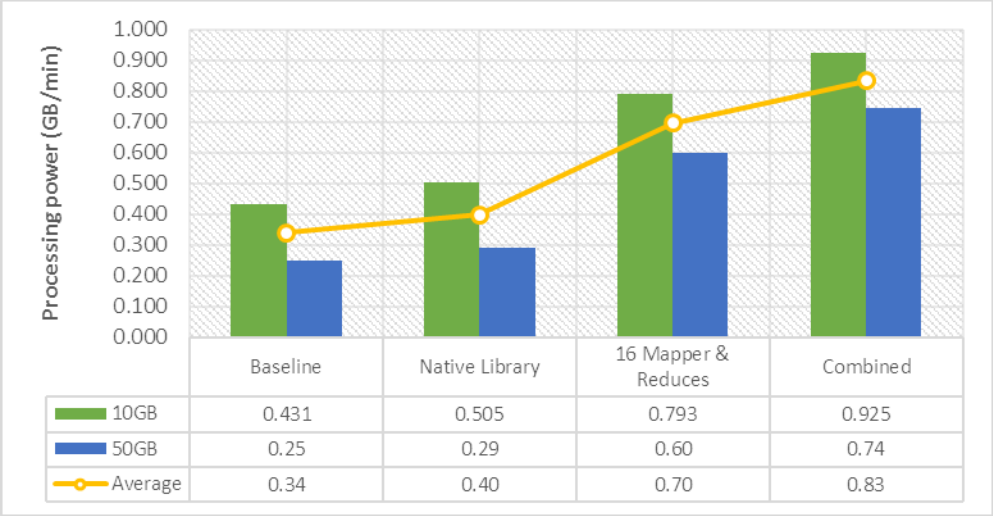


Figure. 4. Using Native Library

By using the native Hadoop Library compiled using the 64bit architecture instead of the built-in Java class, we can see an increase in performance of the cluster Figure 13. This is because the native library communicates directly with the operating system rather than the Java Virtual Machine which reduces the overhead CPU load [3].

By using the native 64bit library we are able to achieve 10% decrease in processing time compared to the baseline. By combining the native library with the optimal configuration from the previous experiment, we are able to achieve more than 50% decrease in processing time.

3.3. HDFS Block Size tuning

HDFS Block size determines the smallest unit of data that the file system can store. The default block size for a spinning hard drive is 4KB. However, since HDFS is meant to handle large files, the default block size for HDFS is 128MB. The size of block size greatly affects Name Node service, the greater the block size, the fewer overhead load for the Name Node service.

The configuration that will be modified is block size of HDFS which is located inside file 'hdfs-size.xml'. The name of the parameter is 'dfs.blocksize' and the value will be the block size in bytes. The block size that will be tested are 64MB (67,108,864 bytes), 128MB (134,217,728 bytes), 256MB (268,435,456 bytes), and 384MB (402,653,184 bytes).

Figure. 5. HDFS Blocksize tuning

Results shown from Figure. 5 provides evidence that the block size of other than 128MB offers no benefit while performing 10GB benchmark.

3.4. Compression codec selection

Hadoop has the ability to compress data at 3 different levels: input, map output, and reduce output. It has also supports 3 codecs that can be used for compression [4]. Depending on the workload, some codecs is better compared to others in terms of compression/decompression speed. Compression helps decrease the network IO overhead with the tradeoff of CPU utilization.

Since TeraSort does not support input data compression and Reduce output compression, in this experiment we can only test Map output compression. The compression codec that will be used in this experiment are gzip, snappy, and bzip2.

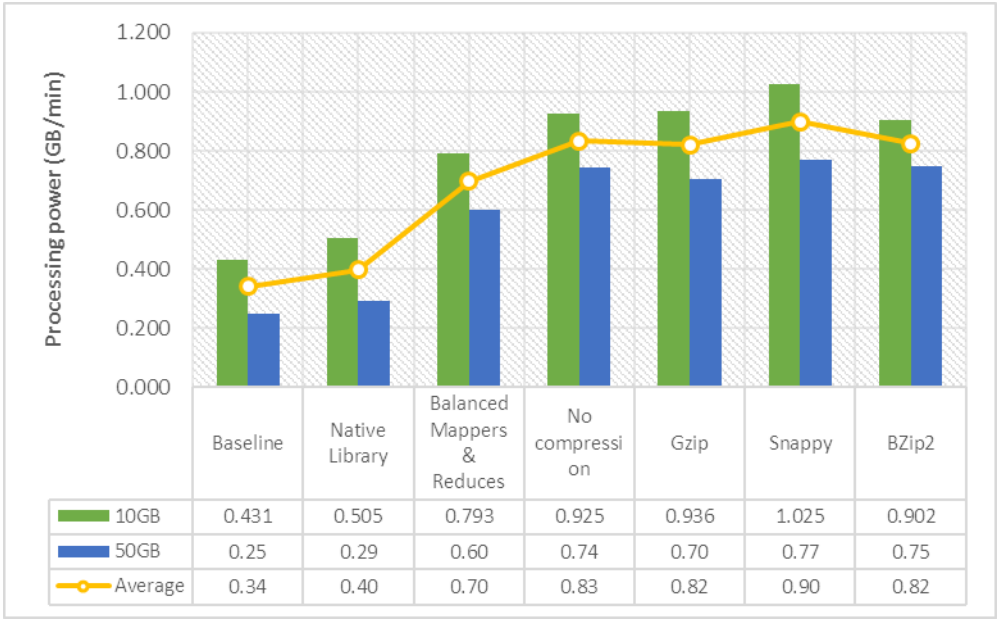


Figure. 6. Compression codec

From the results of Figure. 6 which shows the effect of Map output using compression with different codec. GZip compression offers little to no improvement compared with the baseline while BZip2 offers no benefit but increases the processing time by 4%. This is because BZip2 is able to give better compression ratio than GZip hence the increase in CPU usage slowing the processing time.
Snappy however shows better results than both GZip and BZip2 compression because the library itself aims for the fastest speed yet having good compression ratio. Therefore, we decided to proceed using Snappy as our compression codec as it is the best choice based on the benchmark results.

4. Conclusions

From the research and experiments that we conducted we can conclude that the performance improvement through the editing of configuration is successful. From the initial baseline configuration, we are able to reduce the processing time by 50%. A big improvement gain is contributed by increasing the number of mappers and reduces count. Nevertheless, all improvement gained from other configuration changes and tweaks contributes to the overall performance improvement to the cluster.

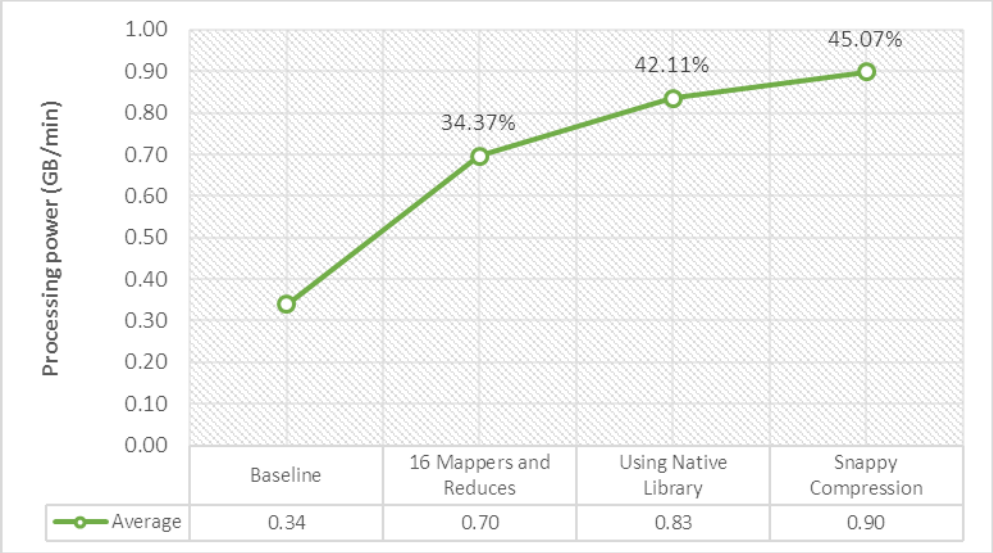


Figure. 7. Overall Improvements

However, not all configuration changes have significant impact on the performance. For example, the HDFS block size configuration change has negative impact on the performance if we try to increase or decrease the block size. This may be contributed by the small test data that we used; if we were to use larger data for the benchmark, the improvement for this configuration can be better.

It is also being proven that using the correct library architecture is essential to improve the performance of the cluster. By default, apache only releases the 32bit architecture version of Hadoop in its downloads page. Therefore, if we are using the 64bit version of Operating System, we need to recompile the entire Hadoop source in order to support the native library support.

Finally, we need to mention that the configurations that we used in this research may only work with our cluster setup and hardware specifications. The results can vary between different systems and cluster setup.

5. Future Works

5.1. Further experiments

Due to the limitations and lack of time, there are different configurations that have been left for the future. There are still hundreds of different configuration parameters that can be tweaked to improve the performance of the cluster. Other than the Hadoop configuration, it is also possible to tweak the Java Virtual Machine to cope with the Hadoop process.

5.2. Hadoop 3

The next major release of Hadoop is version 3. Currently the next version is still in the testing cycle of development at this time of writing this research. It includes number of significant enhancements of features compared with the previous version of Hadoop. Furthermore, Hadoop 3 could offer us new features that previously not available in Hadoop 2. Serval notable new features include: MapReduce task-level native optimization and support for more than 2 NameNodes.

5.3. SSD Storage

Solid State Drive are becoming more affordable as the price declines over the years. It is no way near the price of traditional spinning hard disk just yet, more manufacturers ship their all-in-one computers with SSDs included. One of the main advantage of SSD is their I/O speed compared to mechanical disk. Creating a HDFS using SSD will surely increase the overall performance of the cluster. However, this benefit comes with a risk of disk failure as SSD's lifespan is lower than HDD.

References

Advanced Micro Devices, Inc, "Hadoop Performance Tuning Guide," October 2012. [Online]. Available: https://developer.amd.com/wordpress/media/2012/10/Hadoop_Tuning_Guide-Version5.pdf. [Accessed 21 January 2017].

Fuller, N. C. Meng, M. Li, S. J. Tan, L. Zeng & L. Zhang, (2015.) "Dynamic Resource Allocation in Mapreduce". United States of America Patent 176,635.

Joshi, S. (2012.). "Hadoop Performance Tuning Guide," [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/Hadoop_Tuning_Guide-Version5.pdf.

Ren, Z. W. Jian, S. Wisong, X. Xianghua and Z. Min, (2014). "Workload Analysis, Implications, and Optimization on a Production Hadoop Cluster: A Case Study on Taobao," IEEE Transactions on Services Computing, pp. vol.7, no.2 , pp. 307-321.