2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

Improving Hadoop MapReduce Performance with Data Compression: A Study using Wordcount Job

Kritwara Rattanaopas
Computer Department,
Faculty of Science and Technology,
Songkhla Rajabhat University,
Songkhla, Thailand
kritwara.ra@skru.ac.th

Sureerat Kaewkeeree
Business Computer Department,
Faculty of Management Sciences,
Songkhla Rajabhat University,
Songkhla, Thailand
sureerat.ka@skru.ac.th

***Abstract*—Hadoop cluster is widely used for executing and analyzing a large data like big data. It has MapReduce engine for distributing data to each node in cluster. Compression is a benefit way of Hadoop cluster because it not only can increase space of storage but also improve performance to compute job. Recently, there are some popular Hadoop's compression codecs for example; deflate, gzip, bzip2 and snappy. An over-all compression in MapReduce, Hadoop uses a compressed input file which is gzip and bzip2. This research goal is to improve a computing performance of wordcount job using a different Hadoop compression option. We have 2 scenarios had been test in a study as follows: Scenario I, we use data compression with map output, results found the better execution-time with only snappy and deflate in a raw-text input file. It refers to compression of map output which cans not improve a computing performance than uncompressed. Scenario II, we use a compressed input file with bzip2 with the uncompressed MapReduce that results find a similar execution-time between raw-text and bzip2. It refers to a bzip2 input file can reduce a disk space and keep a computing performance. In concluding, Hadoop compression can investigate the wordcount MapReduce execution-time with a bzip2 input file in Hadoop cluster.**

Keywords-hadoop; mapreduce; compression; wordcount

I. INTRODUCTION

Hadoop is the most popular and powerful tool to analysis a raw big data. It is a distributed system using the MapReduce engine which is the computing paradigm for processing big data sets on a large cluster with a lot of data nodes. File is split into blocks of data and distributes across data nodes in a cluster. It then processes those data in parallel. The Hadoop Distributed File System (HDFS) is a main storage of Hadoop clusters. It is designed to store very large data sets and highly fault-tolerant with a number of replicas of a file on data node. Therefore, Hadoop can access and process data only on HDFS storage. Data size on storage will increased because a number of replication.

This paper focuses on the compression codecs that are available on Hadoop cluster, which are gzip, deflate, bzip2, lz4 and snappy by configuring the codec properties in an xml configuration file. Most researchers aim to improve performance. Yanpei Chen's [1] research described about how to tradeoffs I/O resources (e.g. Storage, Disk IO and Network bandwidth) and energy with compression codecs. However, compression codecs provide some benefit on MapReduce process which includes Map, Sort-Shuffle and Reduce. The native compression codecs can investigate to map output and reducer output. We were purpose the better compression codecs and the best way to use compression codecs in MapReduce process from our study.

The remainder of this paper is organized as follows. Background review and related works include related work and background information (e.g. Hadoop and compression). The MapReduce process with compression codecs are describe in our methodology. In evaluation, we compared the execution time of compression codec on MapReduce. In concluding, we discuss performance and reveal possible future works.

978-1-5386-0449-6/17/$31.00 ©2017 IEEE

II. BACKGROUND REVIEW AND RELATED WORK

A. Related Works

Hadoop is a hot issue of research in high performance computing and cloud nowadays. Most of papers aim to improving and optimizing Hadoop cluster. In Mukhtaj Khan's research [2], he used a Gene expression programming with control Hadoop's parameters (e.g. iosort, mapreduce and input database) those parameters were used in this research. In Andre Wenas's research [3], he used compression (e.g. GZIP, LZJB and ZLE) for file system of data warehouse that the results have a better performance on ZLE. In this paper, we focus on only a native compression (e.g. deflate, gzip, bzip2, lz4 and snnapy). Yanpei Che's research [1] study compare compression with not compression output files with Hadoop benchmark that the results shown a saving energy more than 50%. Jack Li's research [4] uses the Hadoop benchmarks to compare a difference hypervisor on virtualization including Xen, KVM and Commercial hypervisor by using word count, Hive-join tables, Hive-group data and Clustering data. It makes a new idea of using compression method with a large file in our research. In the problem of large small files on HDFS, Chatuporn Vorapongkitipun's research [5] purposes new Hadoop archive method and Zhipeng Gao's research [6] purpose small file merge strategy based LFN method to improve performance of name node in Hadoop cluster. This research had a direct focus on compression in MapReduce with a large file like network traffic log. We purposed a benefit to a compressed input file in this research.

B. Hadoop

Hadoop is the most popular tools in a Big data solution. It is portable suite software and widely used on social network for example, google and yahoo. It's portable over the Java JDK and creates file system as Hadoop Distributed File System (HDFS) which splits large file in blocks and distribute to data node. MapReduce is a main engine of Hadoop. In Hadoop Ecosystem, it has a lot of software built on top. For example Hive is relational warehouse, R Connectors for Statistic function with R language and Mahout is machine learning in Figure 1.

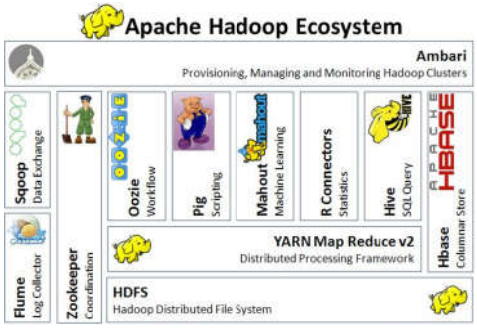

Figure 1. Hadoop Ecosystem [7]

C. Compression

In Linux and Windows operation system, they have a native compression algorithm which can compress and decompress data in storage and disk I/O. For example, gzip and deflate are the same general compression using zlib format. LZ777 concept is not only an original idea of snappy but also a standard of gzip and 'deflate' compression. Also, snappy use Google based and design for Hadoop Ecosystem. In addition, bzip2 is an only compression codec which can split file format. This research interests on using native compressions of a Hadoop tool as follows:

a) gzip: Jean-loup Gailly is the primary developer of gzip. Mark Adler is a developer of gzip and the original developer of Zip [8]. Gzip method uses zlib data format which is portable across platforms. Unlike LZW compression method . It is a native UNIX compression and GIF image format. It has cost in compression. However, the 'deflate' compression method use zlib, same as gzip and Zip, which is default method of Hadoop compression codec. Gzip memory footprint can be reduced data size because it is independent from an input data and never expands data.

b) bzip2: Bzip2 [9] is high quality data compression codec and uses libbzip2 with open-source (BSD-style) license. It uses the available techniques like PPM family. Bzip2 offer performance around twice of compression time and six time faster at decompression in version 1.0.6 which is useful for your overfull disk drives, distribution CDs, backup taps and USB sticks. It can be reduced download times with long distance network. In data recovery mode, it can be restored the compression data and decompressed those parts of the file which are undamaged. Libbzip2 is used to directly read and write .bz2 files with compressed data in memory.

c) Snappy: Snappy [10] is previous name as Zippy. It developed in C++ by Google based with LZ77 concept in 2011. It aims to for very high speed to compression and decompression. The snappy benchmarks use Core i7 with only a single core which 64-bits mode. Its compression ratio is 20-100% lower than gzip. In Hadoop cluster, snappy is the most popular compression codec used in open-source Google project like Cassandra, Hadoop, LevelDB, MongoDB, RocksDB, Lucene.

d) LZ4: LZ4 [11] has a LZ4 library as an open source software using a BSD license. This algorithm can provide speed at 400 MB/s per core (0.16 Bytes/cycle) and has an extremely fast decoder. It has LZ4_HC which is high compression derivative with trading CPU time. This compression is useful with video games.

III. METHODOLOGY

A. Experimental Setup

We employed Hadoop cluster with virtual machine on full virtualization of KVM hypervisor. This cluster was included 1-name node and 4-data node virtual machines over 3 desktop computers with Intel i7-2600 Quad core @ 3.40 GHz which support Intel-VT, 4 GB DDR3, 1.0 TB SATA disks shown in Figure 2. Our Hadoop cluster architecture was described as follows:

- Master Node employed only 1 big virtual machine in Host01 with 4 vCPU and 4 GB RAM
- Data Node involved 2 virtual machines on each physical Host with 1 vCPU and 2 GB RAM

In Software components of physical Host machine, CentOS 7.2 64 bits with KVM virtualization software were installed. The image of virtual machines stored on an extended XFS partition in home directory. All of virtual machine installed CentOS 7.2 64 bits, Java JDK version 1.8 and Hadoop version 2.6.4.

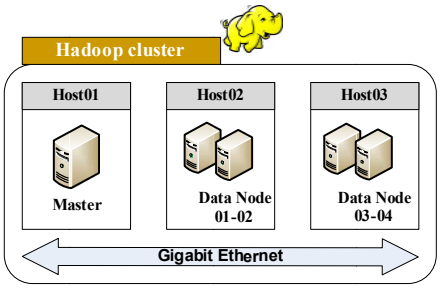

Figure 2. Hadoop cluster Architecture

MapReduce process can use to setup the experiment compression codec with Map output and Reduce output. We configure in "mapred-site.xml" with properties as follows.

1) Map output compression is number-2 in Figure 3

- *"mapreduce.map.output.compress.codec"* has the value parameter which is DefaultCodec, GzipCodec, BZip2Codec, Lz4Codec and SnappyCodec.
- *"mapreduce.map.output.compress"* can use between true or false.

2) Reduce output compression is number-3 in Figure 3

- *"mapreduce.output.fileoutputformat.compress.codec"* has the value parameter which is DefaultCodec, GzipCodec, BZip2Codec, Lz4Codec and SnappyCodec.
- *"mapreduce.output.fileoutputformat.compress"* can use between true or false.

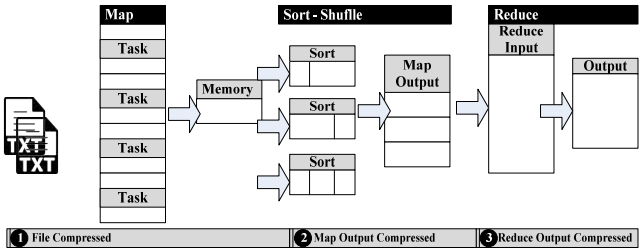

Figure 3. Data compression processing in Hadoop MapReduce [12]

B. Workload

Our experiment scenarios, we conducted the environment to compare an execution time of Hadoop's wordcount MapReduce on virtualization of KVM hypervisor. We used data compression codec with MapReduce process only in Map output. In wordcount MapReduce, its output has a small file size because it contained only a frequency of each word shown in Figure 4. In Figure 3, number-1 used a compression input file which was a secondary hypothesis of this research, evaluating between a raw file and a compressed file.

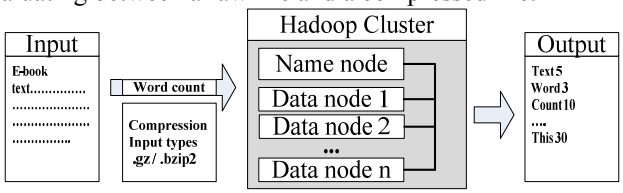

Figure 4. Hadoop's wordcount MapReduce

The workload of our experiment was an e-book text from "Project Gutenberg" [13]. We created the workloads which including 4.8 GB, 7.2 GB, 9.6 GB and 14.4 GB. After created raw-text workload, we used gzip and bzip2 to compress them. The size of the comparison workloads between compressed file and raw-text file show in Figure 5.

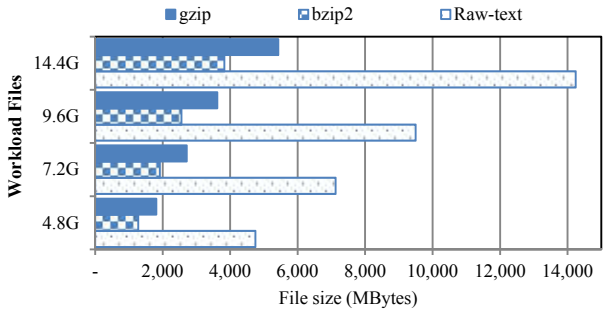

Figure 5. Comparison size of workload files

In Figure 5, a raw-text was actual size of e-book file which shown in x axis and y axis shows the workload files group by compression codec. All each workload's size with bzip2 compression could be compressed at 27% of raw-text in middle bar in each group. For example, case of 14.4 GB (14,242.34 MB) was compressed to 3,828.84 MB (26.89%). In gzip compression, it could be compressed at 38% of raw-text in top bar of each group. For example, case of 14.4 GB (14,242.34 MB) was compressed to 5,425.01 MB (38.09%). This graph in Figure 5 refers to execution's time of bzip2 and gzip which are smaller than raw-text over 60%. We used all of files to evaluate the execution-time in Hadoop's wordcount MapReduce. This research investigated to use a data compress with word count MapReduce job as follows.

- *Scenario I (map output compression):* Input's file of our experiment was only a 4.8 GB workload which is raw-file, gzip and bzip2. In MapReduce process, we used a compressed map output that its compression codec was Default, Gzip, BZip2, LZ4 and Snappy respectively. In MapReduce command, we set map task parameter to 4 and 16 tasks to expand job.
- *Scenario II (input file compression):* Our experiment used a variety of input's size which was 4.8 GB, 7.2 GB, 9.6 GB and 14.4 GB. In MapReduce process, we used an uncompressed both map and reduce process. We set task parameter to 4 and 16 same as scenario I.

All Scenarios, we used only 4 data node and 1 master node with 3-replication. In results, a comparison of an average word count execution-time on each task and compression codec are presented.

IV. Evaluation

We conducted experiments to investigate the computing performance of compression in map output and an input file. We run our workload only a 4.8 GB input file in scenario I and a variety size of input file in scenario II on the Hadoop cluster in KVM hypervisors. In our first experiment, we showed a computing performance in term of execution-time on each workload file types in Figure 6. In next scenario, the results of the execution-time between a raw-text and a bzip2 input files presented as bar graph shown in Figure 7.

A. Scenario I (map output compression)

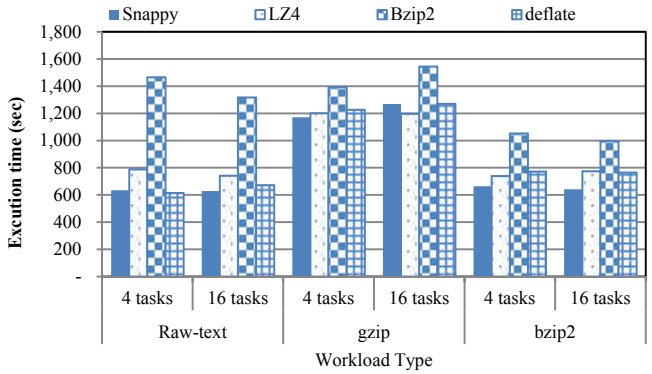

Figure 6. Comparison of word count execution-time with map output compression

In Figure 6, we showed three of main file types which were raw-text, gzip and bzip2. Those were two groups of 4 and 16 tasks. Raw-text input file had a better computing performance than other compression files. It had an execution-

time at 634.67 seconds with snappy as same as deflate at 614.00 second. In contrast, with bzip2, it uses 1,465.75 seconds when is lower than a bzip2 input file at 1,052.00 seconds. These results in all file types have a quite similar execution-time between 4 and 16 tasks of map. In gzip compression, results showed a high execution-time because gzip compression algorithm does not have splittable feature which can compute a split file. It can also compute in only one node in cluster and compatible with a large single node.

B. *Scenario II (input file compression)*

A variety size of input file is 4.8 GB, 7.2 GB, 9.6 GB and 14.4 GB in Figure 7. Raw-text and bzip2 have a similar character to distribute file into all data node in Figure 2. The results show a comparison of execution-time between raw-text and bzip2 group by input file size in Figure 7.

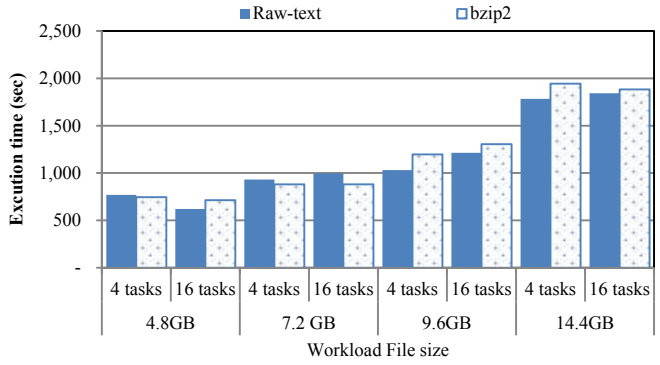

Figure 7. Comparison of word count execution-time between raw-text and bzip2 compression

In Figure 7, the results increased an execution-time along with size of input file. For example, case of raw-text 4.8 GB (16 tasks) had an execution-time at 620.75 seconds which was a half of a raw-text 9.6 GB (16 tasks) at 1,214.43 second. In case of bzip2 file 7.2 GB (16 tasks), it had a similar execution time at 881.50 seconds which was a half of a case of bzip2 file 14.4 GB (16 tasks) at 1,883.50 seconds. These results refer to a linear execution-time in the size of input file. It gives a similar execution-time between raw-text and bzip2 file.

V. Results and Discussion

Our experiment on first scenario, we showed a comparison of compression codecs in Hadoop cluster which were deflate, gzip, LZ4 and snappy. The better performance found in only map output compressed of deflate and snappy with raw-text input. First hypothesis related to number of map tasks in MapReduce. These results in Figure 6 and Figure 7, it gave a similar computing performance. The compression tradeoff with CPU usage, results could refer to compression time in MapReduce process in scenario I. The compressed input file must be tradeoff CPU time to decompressed that before the split files transfer and compute in data node. On the other hand, only map output with bzip2 had a better execution-time than raw-text but its execution-times were not lowering than raw-text which shown as others map output compression. However, scenario II's results in Figure 7 could refer to the compression tradeoff with CPU time for a bzip2 input file. In bzip2 compression, we could save HDFS storage space in 3-replication more than 11×3 GB in case of 14.4 GB with a few seconds of CPU time for decompression. The results of 14.4 GB find raw-text's execution-time at 1,842.00 seconds lower than bzip2's execution-time at 1,883.50 seconds only 53.50 seconds that execution-time can refer to a saving space of HDFS up to number of replication.

VI. Conclusions

In conclusion, we purposed a data compression with MapReduce in Hadoop cluster. The bzip2 input files are compatible to wordcount job like raw-text file. It can save storage space more than 70% of raw-text file. Hadoop benchmarks have other application for a large output file of reduce phase which is not focused on this research. However, Hadoop has Hive-warehouse and HBase for database execution that compression codec can be applied to them before store data to HDFS.

Acknowledgment

The authors would like to acknowledge Faculty of Science and Technology, Songkhla Rajabhat University, Thailand for support of this research.

References

[1] Y. Chen, A. Ganapathi, R. H. Katz, "To Compress or Not To Compress Compute vs. IO Tradeoffs for MapReduce Energy Efficiency," *Green Networking 2010*, New Delhi, India, pp. 23-28, August 30, 2010.

[2] M. Khan, Z. Huang, M. Li, G. A. Taylor, and M. Khan, "Optimizing hadoop parameter settings with gene expression programming guided PSO," *Concurrency Computat.: Pract. Exper.*, vol. 29, no. 3, p. n/a-n/a, 2017.

[3] A. Wenas, S. Suharjito, "Improving Data Technology Warehouse Performance Using Filesystem with GZIP, LZJB and ZLE Compression," *Jurnal Informatika dan Sistem Informasi*, vol. 2, no. 2, pp. 41–51, Feb. 2017.

[4] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, and C. Pu, "Performance Overhead among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks," in *2013 IEEE International Congress on Big Data*, pp. 9–16, 2013.

[5] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," in *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2014, pp. 200–205.

[6] Z. Gao, Y. Qin, and K. Niu, "An effective merge strategy based hierarchy for improving small file problem on HDFS," in 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2016, pp. 327–331.

[7] The big data blog, "Hadoop Ecosystem Overview", [Online] Available: http://thebigdatablog.weebly.com/blog/the-hadoop-ecosystem-overview, 2016.

[8] A Massively Spiffy Yet Delicately Unobtrusive Compression Library, Inc., "zlib", [Online] Available: http://www.zlib.net/, 2017.

[9] CVE-2010-0405, "bzip2 and libbzip2", [Online] Available: http://www.bzip.org/index.html, 2017.

[10] Wikipedia, the free encyclopedia, "Snappy (compression)", [Online] Available: https://en.wikipedia.org/wiki/Snappy_(compression), 2017.

[11] Takayuki Matsuoka, "LZ4", [Online] Available: http://lz4.github.io/lz4/, 2017.

[12] G. Kamat, S. Singh, "Compression Options in Hadoop - A Tale of Tradeoffs", [Online] Available: https://www.slideshare.net/Hadoop_Summit/kamat-singh-june27425pmroom210cv2, 2013.

[13] Project Gutenberg, "Free ebooks by Project Gutenberg", [Online] Available: https://www.gutenberg.org/, 2017.