

11-2016

Performance Comparison of a Hadoop DFS to a Centralized File System of a Single Machine

Zhao Xie
mndarren@gmail.com

Follow this and additional works at: https://repository.stcloudstate.edu/csit_etds

Recommended Citation

Xie, Zhao, "Performance Comparison of a Hadoop DFS to a Centralized File System of a Single Machine" (2016). *Culminating Projects in Computer Science and Information Technology*. 14.
https://repository.stcloudstate.edu/csit_etds/14

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

**Performance Comparison of a Hadoop DFS to a Centralized File System of a
Single Machine**

by

Zhao Xie

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science in

Computer Science

December, 2016

Starred Paper Committee:
Donald Hannes, Chairperson
Jie Meichsner
Dennis Guster

Abstract

With the coming of a big data era, Hadoop, developed by Doug Cutting and Mike Cafarella, was presented in 2005 [1], which turned over a new page in the history of cloud computing. The Hadoop Distributed File System (HDFS) is one of the most fundamental layers in Hadoop. In the big data world, the performance of dealing with big data from HDFS cannot satisfy the need because the amount of big data is getting larger and larger, and simultaneously, the increasing rate of growth of big data is faster and faster. Nowadays various new distributed file systems (DFS) are published attempting to solve this issue. The core problem hindering the performance from becoming more effective is the metadata service layer in HDFS, and most of the new DFSs are focusing on improving the metadata service as well.

Most of the above-mentioned cases are centering on the issue of solving the big data problem. However, for a small or medium-sized company, the data they may use is not so big. In this case, do they need to build a distributed system to deal with their data? Of course, the data in these companies will be getting larger and larger. When will be the best time for them to need a distributed system to manage their data? This paper attempts to address this problem by comparing the different performances between a distributed system computation and a serial computation.

Acknowledgement

I would like to thank my advisor Dr. Hamnes for offering a lot of valuable suggestions to my work. Without his guidance and assistance, it could not have been possible that my whole progress has gone so smoothly. Also, I would express my deep appreciation and indebtedness to the committee members—Dr. Meichsner and Dr. Guster, who contributed their time and energy in modifying my paper and providing insightful suggestions. My sincere appreciation also goes to Martin Smith, who provided great support in helping me set up the hardware and system environment for the laboratory work. Finally, my family, Hailei and Monica, gave me a huge support for this research.

Table of Contents

	Page
List of Tables	5
List of Figures	6
Section	
1. Introduction	7
2. Background	7
3. Problem Issue	8
4. Methodology	9
4.1 Laboratory Environment	10
4.2 Laboratory Network Deployment	11
4.3 Algorithm Choice and Design of Data File Sizes	12
4.4 Program Execution	13
5. Result Data Analysis	23
6. Conclusion	25
7. Limitation	25
References	27
Appendices	
A. Hadoop 2.6.0 Installation on Ubuntu 14.04	30
B. Centralized File System Performance Test	49
C. Hadoop DFS Performance Test	51
D. Source Codes	59

List of Tables

Table	Page
1. Hardware Configuration	10
2. Virtual Machine Configuration	11
3. Software Configuration	11
4. Selected Data File Sizes	13
5. Single Machine Result Data	18
6. Hadoop Cluster Tested Result Data	22
7. Total Execution Time Comparison	23

List of Figures

Figure	Page
1. The Hadoop network deployment	12
2. External sort step 1 and step 2	15
3. External sort step 2 and step 4	16
4. Hadoop sort data flow	20
5. Hadoop cluster and single machine total execution time comparison	24
6. The ratio of total execution time between Hadoop cluster and single machine	25

1. Introduction

All of the existing distributed systems, such as HDFS from Hadoop [2], GFS from Google [3], TFS from Alibaba, BWFS from Chinese researcher [4], and Ceph [5], focus on dealing with big data or biggest data. Other researchers provided many new and efficient approaches to enhance the distributed systems. All of these approaches, such as CHMasters (Consistent Hashing Masters) [6], DROP (Dynamic Ring Online Partitioning) [7], and CEFLS (Cost-Effective File Lookup Service) [8], are focusing on the improvement of the metadata service, whose basic purpose is still managing the big data or biggest data as well. The purpose of this paper is to compare differences in performance between a distributed system computation and a serial computation in dealing with bigger data. The author simulates a distributed system by using the Hadoop framework. After building up this Hadoop cluster, a series of lab experiments are performed.

2. Background

For Big Data, there are three important history time points: First, in October 1997, Michael Cox and David Ellsworth published an article, named “Application-controlled demand paging for out-of-core visualization”, in the *Proceedings of the IEEE 8th conference on Visualization*. It is the first article in the ACM digital library to use the term “big data” [9]. From that point on big data started becoming a problem. Second, in 2008, a number of prominent American computer scientists popularized the term, predicting that “big-data computing” will “transform the activities of companies, scientific researchers, medical practitioners, and our nation’s defense and intelligence operations” [10]. Big data developed a real big problem. Third, recently (in 2012) the total amount of “big data” reached 7.9 ZB, and

will touch 35ZB in 2020 [11]. Every day we create 2.5 quintillion bytes of data [12]. Now big data is an important and popular problem.

For distributed systems, there are two important time points:

First, the article, “The Google File System”, was published in October 2003 [3]. This is an important milestone in distributed system development, because this is the first time for the distributed file system, as a best approach to solve big data problem, to have been successfully applied in a large company.

Second, Hadoop was produced in 2005 by Doug Cutting and Mike Cafarella, the first open source distributed system framework. Subsequently, Amazon launched its Amazon Web Services (AWS) in 2006 [13]. IBM Cloud, Microsoft Azure, Yahoo, Alibaba, and other large companies all built their own distributed system.

All of these distributed system technique improvements focus on dealing with big data or biggest data. Therefore, the ability and technology to manage big data is getting stronger and stronger, and the amount of data to be dealt with per second is becoming bigger and bigger.

3. Problem Issue

The research about the big data problem are all aiming at one end point, how to enhance or improve the distributed system techniques to deal with larger amounts of data per second. It was hard for me to find any researchers involving paying attention to another end point: how much data should an enterprise accumulate before it starts to think about creating a distributed system to process their data.

Most large companies grew from a small one, but does the company need to build its own distributed system when it's setting up? The answer is definitely "no" in that usually when a company starts to run, there are insufficient resources, such as money, human resources, etc. to support this, and also because there is not any big data that needs to be dealt with. So when should the company consider building its distributed system to handle its data? In other words, the data is increasing as the company is expanding. How much data should the company accumulate before it starts to think about creating a distributed system or borrow one to handle it? In this paper, I attempt to search for an appropriate answer by performing a series of lab experiments.

4. Methodology

A number of components will be examined in this study. It begins with efforts in building up a distributed system by using the Hadoop framework, and then creating a set of files as data or "big data" to be tested. After that, the Word Count program, a Hadoop example program, will be executed to examine whether the new Hadoop cluster works well. Appendix A shows the steps in the installation of Ubuntu and Hadoop to create the test environment.

The second step is to choose an algorithm for comparison purposes. The algorithm selected was the sorting of data files of a fixed size. In this step, the specific sizes of data files were selected. According to the chosen algorithm and the hardware and software configuration, the third step is to write programs, including a Hadoop sort program, internal sort program, external sort program, and create a data generation program and a program to check the result data file (all of the source code files are located in Appendix D).

After having finished writing all of the programs, the fourth step is to perform the program execution and to collect the resulting data. The whole lab experiments are divided into two main parts: single machine test and Hadoop cluster test. Both of these tests are performed on the same big size files so that the resulting data will be comparable. Appendix B shows the results of performing the test on the centralized file system of a single machine. Appendix C shows “tuning” of parameters—adjustment of Hadoop parameters for better performance.

Eventually, the collected data will be compared and analyzed, and then conclusions will be drawn.

4.1 Laboratory Environment

Tables 1-3 show the configuration of the Hardware, VMs and Software used in this study. All of the Virtual Machines were located on one powerful physical server. Table 1 shows the server hardware configuration.

Table 1

Hardware Configuration

Hardware Name	Pattern
CPU	Intel Xeon E5-2680 (2.8 GHz, 10 physical cores)
Memory	DDR3 (256 GB)
Hard Disk	A centralized storage device (SAN server)
Network Bandwidth	2GB

Table 2
Virtual Machine Configuration

Hardware Name	Information
Virtualization Platform	Version: VMware vSphere 5.5
CPU	Intel ® Xeon ® CPU E5-2680 V2 @ 2.80GHz, 2 cores
Memory	Size: 4 GB
Hard Disk	Size: 124 GB
zaho-hadoop1	IP: 10.59.7.42
zaho-hadoop2	IP: 10.59.7.43
zaho-hadoop3	IP: 10.59.7.44
zaho-hadoop4	IP: 10.59.7.45
zaho-hadoop5	IP: 10.59.7.46
zaho-hadoop6	IP: 10.59.7.47
zaho-hadoop7	IP: 10.59.7.48
zaho-hadoop8	IP: 10.59.7.49

Table 3
Software Configuration

Software Name	Version
Operating System	Ubuntu 14.04.2, X86_64
Java Development Kit	JDK 1.7.0_79
Secure Shell	OpenSSH_6.6.1p1
Hadoop	Hadoop 2.6.0

4.2 Laboratory Network Deployment

The Hadoop framework is deployed on 8 virtual machines as shown in Table 2 Virtual Machine configuration. The master node is set up on the VM with IP address 10.59.7.42,

while the other 7 VMs with IP addresses 10.59.7.43 ~ 10.59.7.49 are slave nodes. For the Hadoop framework, namenode service and YARN component are launched on the master node, and datanode service is set up on each slave node. For single machine tests, either internal sort or external sort, all will be executed on the VM with IP 10.59.7.43. When all single machine tests are running, the whole Hadoop service is stopped so that the tests are able to fully utilize the physical resources. Another reason to choose that VM, and not the master VM, is that if the datanode service files were damaged because of single machine tests, the node can be easily fixed by copying those files from other slave nodes. Figure 1 summarizes the Network Deployment.

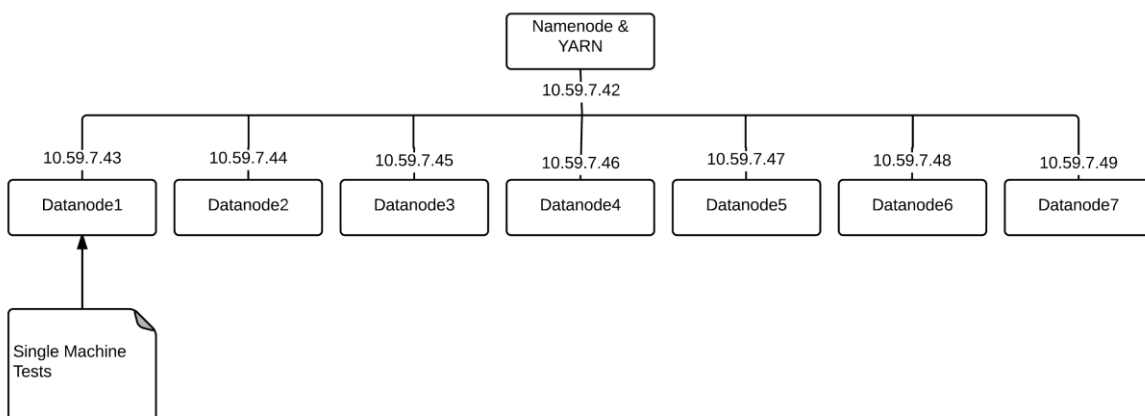


Figure 1. The Hadoop network deployment.

4.3 Algorithm Choice and Design of Data File Sizes

This project chose quick sort as the test algorithm because Hadoop uses built-in quick sort in the MapReduce framework so that the sorted result data information will be comparable. According to the hard disk size (124 GB) and the memory size (4 GB) limitation, the largest data file size to be tested can be about 40 GB. This is due to temporary file space needs and OS needs. Therefore, two cases are chosen for internal sort, and another six cases

for external sort as shown in the following table. In order to keep the data comparable, the maximum value of integers in each input data file is 100 million, and the input data file for each program execution is created as one text file, not multiple files.

Table 4
Selected Data File Sizes

Case No.	1	2	3	4	5	6	7	8
Number of Integers(million)	130	170	450	900	1700	2800	3500	4600
Data File Size (GB)	1.16	1.51	4	8	15.11	24.89	31.11	40.89

4.4 Program Execution

There are two parts in this section: Single machine test and Hadoop cluster test. For each part, I will provide program description, data process approach and final result data.

4.4.1 Single machine test

4.4.1.1 Program descriptions

In single machine test, four programs are needed: one is to create data as input; another is internal sort program to test the files whose size will be less than free memory size; the third is external sort program to test the files whose size will be larger than free memory size; the last is the check output program to check if the output is correct.

In order to test the performance of the file system by using the sort algorithm, what we need first is the input data file. The program, CreateData.java, is to create data files as input by which we can generate multiple files by modifying the variable NUMOFFILES

(NUMOFFILES was set to 1 for this testing), and we can build files with different sizes by changing the value of variable NUMSPERFILE. The code of the program, CreateData.java, is shown in Source Code 1 in Appendix D.

The internal sort program is to test smaller-size files. This program contains two classes: one is quick sort class; the other is the main sorting class. The code of the program are shown in Source Code 2 and 3: QuickSort.java and SortingData.java.

The external sort program is to test bigger-size files. It is the main program in the single machine test because most of our data files are bigger data files. The external sort program contains one code file (ExternalSort.java) and the code is based on [14].

4.4.1.2 Data process approach

Once the size of unsorted data file exceeds the size of free memory, the internal sort algorithm will not work anymore, in which case we will have to use an external sort algorithm. There are four steps [14]:

Step 1, split the large file into small temporary files, and all these temporary files will be deleted automatically by Java VM before this program stops;

Step 2, sort each of these small temporary files and store them in the disk. This step is executed together with Step 1; Figure 2 shows the work of these two steps.

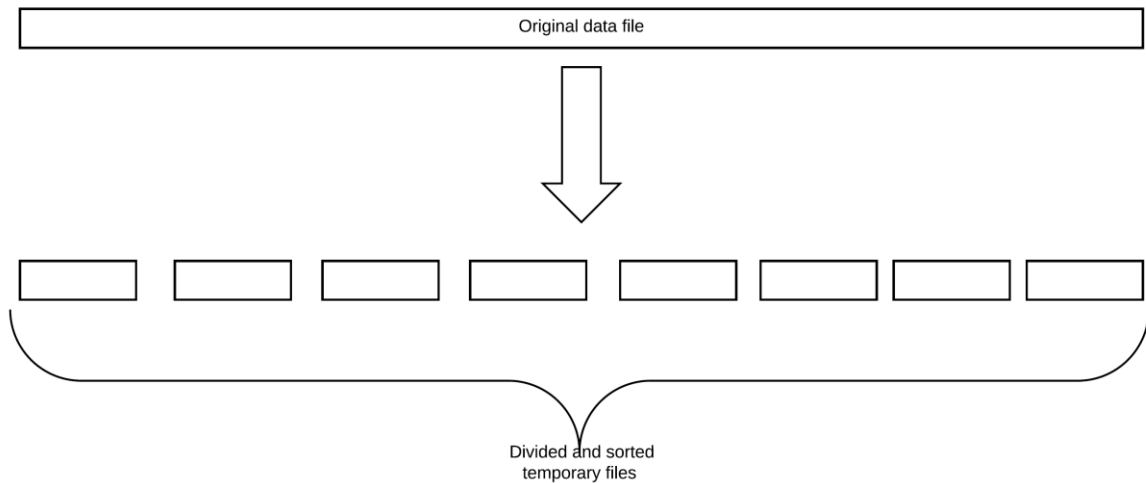


Figure 2. External sort step 1 and step 2.

Step 3. Delete the original data file, the reason for which is to save enough space to store the final sorted data file.

Step 4, Merge all sorted small files into one sorted file. This step will be completed via a priority queue and a certain number of buffers and all buffers are pushed in this priority queue. Each buffer connects to one temporary sorted file. For every buffer, once its integers are used up, it will be automatically reloaded from its connected temporary file. The program will take the first item, which is always the smallest integer in the queue, out of the first buffer and write it into the final-result data file.

This process proceeds repetitively until all the integers are written into the final-result data file. For more details involving the work described in Step3 and 4, you could refer to Figure 3 as below:

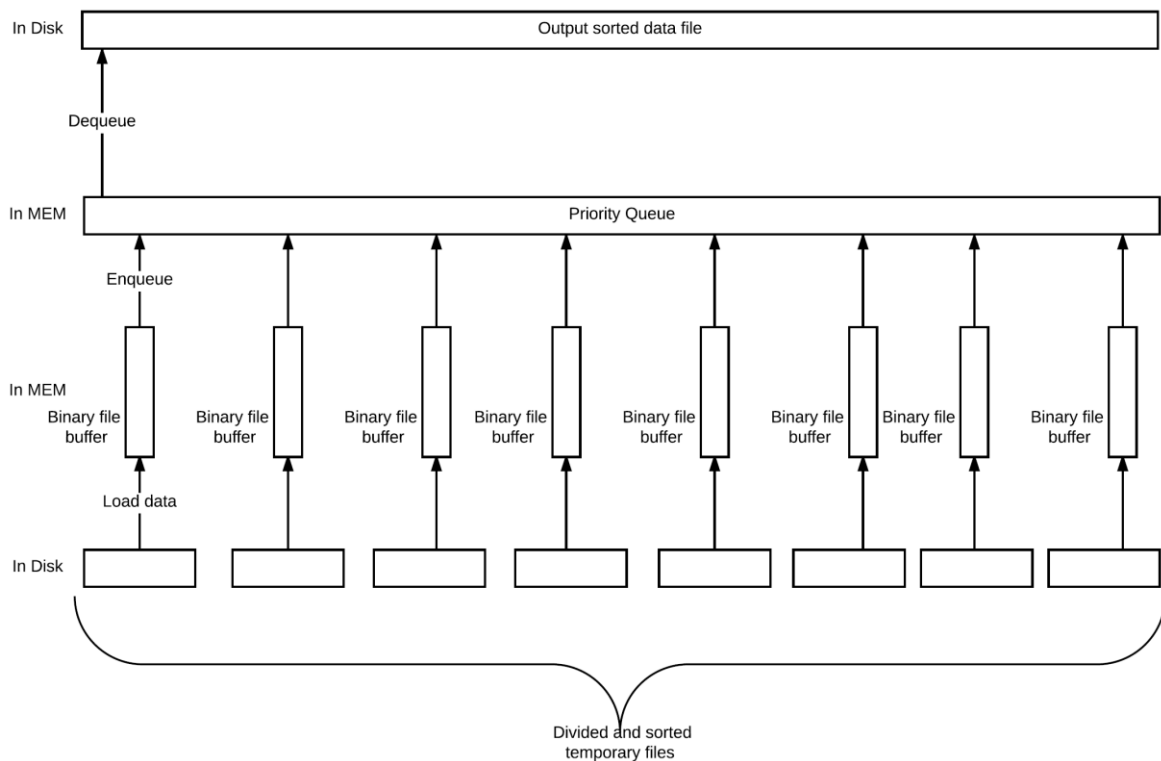


Figure 3. External sort step 3 and step 4.

The external sort program code (ExternalSort.java) is based on [14].

4.4.1.3 Result data from single machine test

All the tests were launched at least three times. The average result data will be used as the comparison data. Timers were added to the sort programs to measure the amount of time spent in reading, writing, and sorting. Also, it will provide the total execution time. However, this measurement itself also took some CPU time, and based on my calculation, it turned out that it had increased the overall run time at about 0.59% from a version without these timers.

In the Internal sort algorithm, I employed the quick sort algorithm, $O(n \log n)$. In the External sort algorithm, I adopted the built-in java function, `Collections.sort()` in Step 2, which is an optimized merge sort [15], so the time complexity $O(n \log n)$ is the same with that for quick sort.

In the Internal sort algorithm, the program reads data as int type from the disk, while in the External sort algorithm, the program reads data as a string type from the disk. Since Hadoop MapReduce program reads data as strings from disk, and then parses these data from characters to integers, the External sort program and the MapReduce program are comparable.

All these programs, including Internal sort program, External sort program, and MapReduce program are written in Java, which makes the results more comparable.

The following is the collected result data.

Table 5
Single Machine Result Data

Data Collection Results																
Test No.	Algorithm	Files	Max Value (M)	Nums PerFile (M)	FileSize(B)	FileSize (MB)	ReadTime (minutes)	Read Throughput (MB/Sec)	WriteTime (minutes)	WriteThroughput (MB/Sec)	SortTime (minutes)	TotalTime (minutes)				
												Low	Medium	High	Average	Δ
1	Internal sort	1	100	130	115554455	1155.55	1.03	18.76	0.73	26.25	2.46	2.67	4.97	5.01	4.21	1.34
2	Internal sort	1	100	170	1511110156	1511.11	3.10	8.13	1.43	17.63	3.36	6.66	6.96	10.04	6.81	0.21
3	External sort	1	100	450	4000001286	4000	4.47	14.90	3.53	18.88	14.90	20.84	22.65	25.22	21.75	1.28
4	External sort	1	100	900	799989627	7999.99	4.51	29.54	7.09	18.79	33.76	43.20	43.94	48.96	43.57	0.53
5	External sort	1	100	1700	15111135465	15111.14	9.94	25.35	14.25	17.68	70.87	89.49	91.23	104.43	90.36	1.23
6	External sort	1	100	2800	24888854625	24888.85	20.44	20.29	23.16	17.91	116.11	156.44	160.71	162.00	161.36	0.91
7	External sort	1	100	3500	31111123081	31111.12	28.83	17.98	25.14	20.63	147.17	189.36	196.05	218.00	192.71	4.73
8	External sort	1	100	4600	40888864416	40888.86	41.11	16.58	30.16	22.60	193.63	250.09	261.24	283.38	255.66	7.88

In Table 5, the first column, “Test No.”, indicates there are altogether eight test cases. The “Algorithm” column shows that the first two test cases use internal sort while the other cases use the external sort. The “Files” column with value 1 means that there is only one input-data file for all the tests. The “MaxValue” column refers to the maximum value of the integers in the input file is 100 million. To make the result comparable, the same maximum value was used. As for “NumsPerFile”, it refers to the number of integers in the input file. The file size was controlled by way of modifying the “NumsPerFile” parameter.

Moreover, timers were set up to record the 4 types of time: “ReadTime”, “WriteTime”, “SortTime” and “TotalTime”. In terms of their relationship, the total time is the sum of read time, write time, and sort time. Besides, dividing “File Size” by “Read Time” is equal to “Read Throughput”. The “Write Throughput” is calculated in a similar manner.

Furthermore, each test case was performed at least three times. In “TotalTime” column, the

“Average” = (Low + Medium + High)/3. $\Delta = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$. This formula is to calculate the standard deviation for the Sample case.

4.4.2 Hadoop cluster test

4.4.2.1 Program description

In the Hadoop cluster test, there are three programs to be used: data-creating program, result-data-checking program, and the Hadoop sort program. The first two are the same as those used for the single machine test. By doing so, it makes sure the input data file and output data file are comparable. For Hadoop sort program, its inner class extends mapper or reducer of the Hadoop MapReduce framework. (For more information, please refer to Source Code 6 in Appendix D.)

4.4.2.2 Data process approach

The following are the steps in how Hadoop processes the data [16]:

Step 1. Put the data file into the HDFS. The HDFS will then split the data file into smaller data pieces and store each piece with key/value format in 3 copies in the HDFS.

Step 2. Map the data. In this phase, the data pieces will be processed for the first time. Usually the important data will be screened out from the original data pieces.

Step 3. Shuffle the data. The data pieces will be shuffled or sorted based on the key of each key/value pair.

Step 4. Reduce the data. In this phase, Hadoop will further process the data pieces according to the command of the MapReduce program.

The whole Hadoop sort data flow process is shown in Figure 4, in which the first row means the steps or phases, the second row introduces the data type changes in each different phase, and the last row provides an example and demonstrates its changes in each phase. For this example, it is assumed that the integers shown in the input box are the smallest five integers so that the change for each phase, especially for output, could be clearly observed.

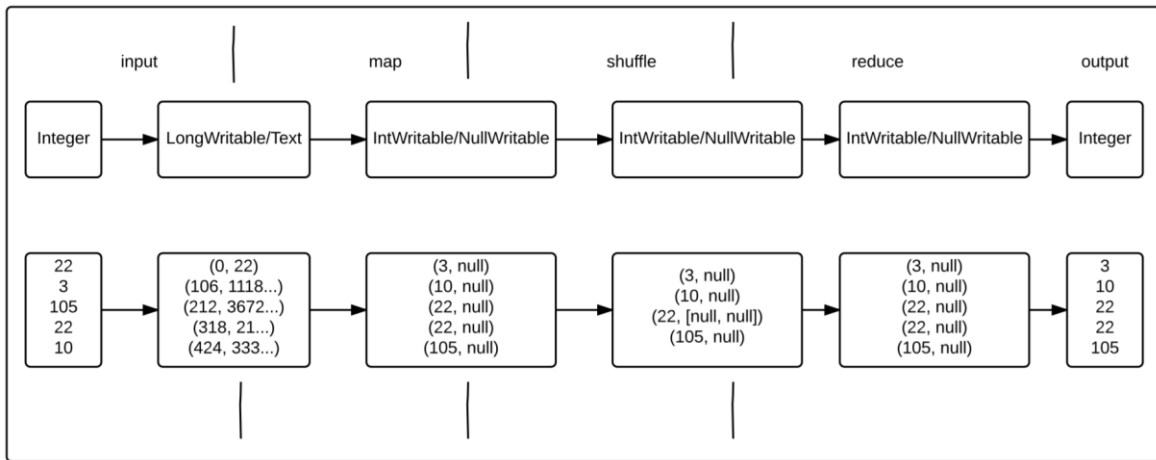


Figure 4. Hadoop sort data flow.

4.4.2.3 Result data from Hadoop cluster test

Similarly, all the Hadoop cluster test cases were launched at least three times. In the Hadoop sort algorithm, the program uses default sort algorithm in the shuffle phase, which is the merge sort [16].

Table 6 below shows the Hadoop cluster result data, in which the column headings are totally the same with those of the Table 5, the single machine result data.

The following is the collected result data.

Table 6
Hadoop Cluster Tested Result Data

Test No.	Algorithm	Files	Max Value (M)	Nums PerFile (M)	FileSize(B)	FileSize (MB)	Hadoop tested result data						TotalTime (minutes)			
							ReadTime (minutes)	Read Throughput (MB/Sec)	Write Time (minutes)	Write Throughput (MB/Sec)	SortTime (minutes)	LOW	Medium	High	Average	Δ
1	Hadoop sort	1	100	130	1155554455	1155.55	0.06	328.73	0.13	145.25	9.86	9.82	10.15	10.18	10.05	0.20
2	Hadoop sort	1	100	170	1511110156	1511.11	0.09	267.29	0.23	108.99	12.37	12.37	12.73	12.98	12.69	0.31
3	Hadoop sort	1	100	450	4000001286	4000	0.15	439.24	0.93	71.48	34.02	34.7	35.37	35.23	35.10	0.35
4	Hadoop sort	1	100	900	799989627	7999.99	0.67	198.56	2.23	59.71	71.65	74.5	74.53	74.63	74.55	0.07
5	Hadoop sort	1	100	1700	15111135465	15111.14	2.02	124.78	4.36	57.78	148.76	153.48	154.95	156.98	155.14	1.76
6	Hadoop sort	1	100	2800	2488854625	24888.85	3.37	123.09	6.10	68.03	237.89	239.28	244.37	258.43	247.36	9.92
7	Hadoop sort	1	100	3500	31111123081	31111.12	4.46	116.14	7.86	65.99	297.10	296.3	314.27	317.7	309.4	11.494
8	Hadoop sort	1	100	4600	40888864416	40888.86	6.58	103.58	10.60	64.28	404.26	419.63	419.75	424.95	421.4	3.0375

5. Result Data Analysis

The total execution time should be the most important measurement for the two parts of the whole experiment in that it probably will answer the question of our problem issue raised at the beginning of this paper. Table 7 indicates the total execution time comparison between Hadoop cluster and the single machine.

Table 7

Total Execution Time Comparison

FileSize (GB)	Hadoop (minutes)	Single (minutes)	Hadoop/Single
1.15	10.05	4.21	2.38
1.51	12.69	6.81	1.86
4	35.10	21.75	1.61
8	74.55	43.57	1.71
15	155.14	90.36	1.72
24.89	247.36	161.36	1.53
31.11	309.42	192.71	1.61
40.89	421.44	255.66	1.65

Figure 5 and Figure 6 show the curves using absolute values and relative values, respectively. To be specific, Figure 5 explicitly indicates that with the enlargement of the size of the input data file, the total execution time of both Hadoop cluster and single machine is increasing simultaneously; however, their respective increasing speeds are different.

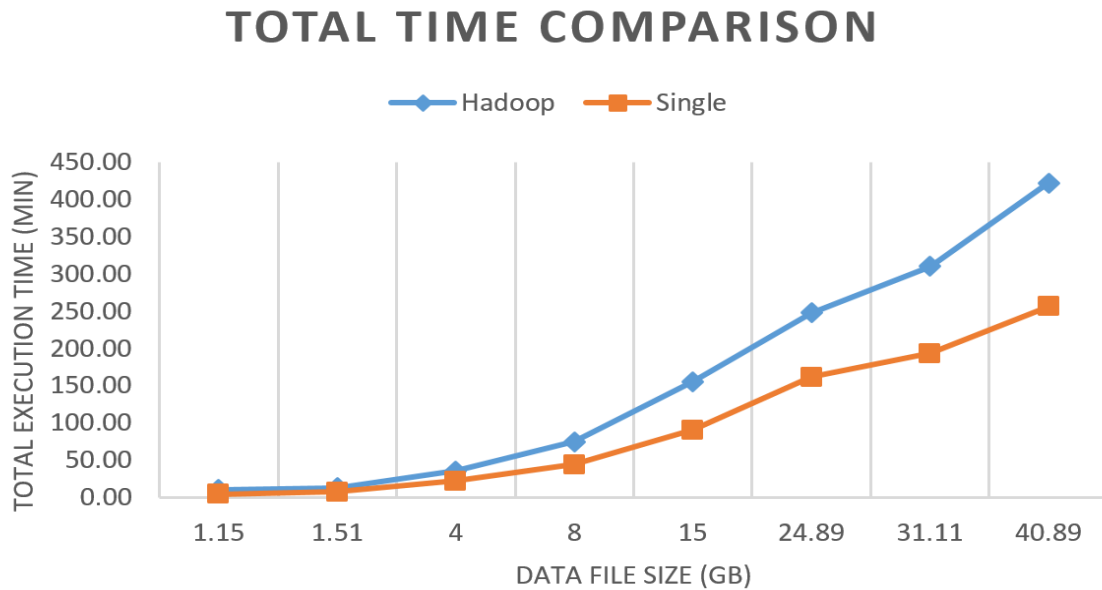


Figure 5. Hadoop cluster and single machine total execution time comparison.

My original conjecture before these lab experiments was that the result ratio curve should be a curve going from top left to the right down and through the horizontal line with value 1. Besides, the intersection point between the curve and the horizontal line with value 1 would be what we are looking for. Before that point, the performance of the single machine would be better than Hadoop cluster and after that point, the performance of the Hadoop cluster would be better than that of a single machine.

However, the result of the whole experiment from Figure 6 indicates that the real result curve is different from my original conjecture. It turns out that the curve does not go through the horizontal line with value 1, but it goes up and down between value 1.5 and value 2.0.

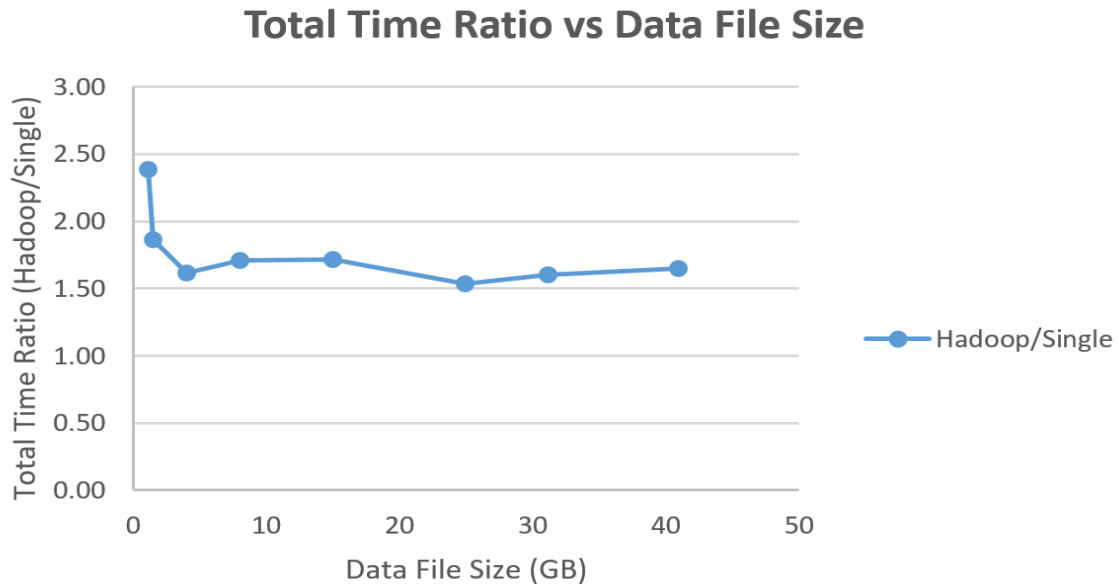


Figure 6: The ratio of total execution time between Hadoop cluster and single machine.

6. Conclusion

According to the previous analysis on the result data, we didn't achieve the expected data size point. At that point, single machine and Hadoop cluster will get same performance; before that point, the performance of single machine is better than that of Hadoop cluster; after that point, the performance of Hadoop cluster is better than that of single machine. With the data size ranging from 1 GB to 40 GB and on the hardware and software configuration used, the single machine always performs better than the Hadoop cluster.

7. Limitation

In all the lab experiments, there exist some limitations, which could possibly be the reasons that explain the disparity between my conjecture and the actual result. Here are the limitations:

- (1) The sizes of the data files in the experiments were still too small. The size of the biggest data file in this research only has 40 GB, while generally only those whose sizes are at least 1 TB can be called Big Data;
- (2) The Hadoop cluster that was used is too small. The Hadoop cluster in this research includes only 8 virtual machines and each machine in the cluster has very limited resources;
- (3) The data structure was too simple. Our input data files only include integers.

In the future, it is possible for researchers to address the above-mentioned issues in their lab experiments. On the condition that these researchers carry out adequate experiments and have big enough data sizes, the critical point probably will be found. This is because a lot of existing research results have indicated that when the cluster is big enough, the performance of a Hadoop cluster is much better than that of a single machine. For example, AliCloud from Alibaba sorted 100 TB of data within 377 seconds on Oct. 28, 2015, breaking the previous world record of 1,406 seconds set by Apache Spark [17]. If this job was done by a single machine, it apparently would expend a lot of time to complete it.

References

- [1] D. Harris. (2013, Mar. 4). "GIGAOM–The history of Hadoop: From 4 nodes to the future of data" [Online]. Available: <https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/>. [Accessed 17 April 2015].
- [2] T. White, "The Hadoop distributed file system," in *Hadoop--Definitive Guide*, Beijing, Cambridge, Farnham, Köln, Sebastopol, Tokyo: O'Reilly Media, Inc., 2011, pp. 41-71.
- [3] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *SOSP (Symposium on Operating Systems Principles)*, Bolton Landing, New York, 2003.
- [4] D. Yang, H. Huang, J. Zhang, and L. Xu, "BWFS: A distributed file system with large capacity, high throughput and high scalability," *Journal of Computer Research and Development*, vol. 42, no. 6, pp. 1028-1033, 2005.
- [5] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *USENIX*, 2006.
- [6] M. Xu, J. Zhou, W. Zhou, and H. An, "CHMasters: A scalable and speed-efficient metadata service in distributed file system," in *Parallel and Distributed Computing, Applications and Technologies*, Gwangju, 2011.
- [7] Q. Xu, R. V. Arumugam, K. L. Yong, and S. Mahadevan, "DROP: Facilitating distributed metadata management in EB-scale storage systems," in *Mass Storage Systems and Technologies*, Long Beach, 2013.
- [8] X. Li, B. Dong, L. Xiao, L. Ruan, and D. Liu, "CEFLS: A cost-effective file lookup service in a distributed metadata file system," in *Cluster, Cloud and Grid Computing*, Ottawa, ON, 2012.
- [9] G. Press. (2013, May 9). "A very short history of big data (Forbes website)" [Online]. Available: <http://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/>. [Accessed 22 April 2015].
- [10] G. Press. (2014, Sep. 3). "12 big data definitions: What's yours? (Forbes website)" [Online]. Available: <http://www.forbes.com/sites/gilpress/2014/09/03/12-big-data-definitions-whats-yours/>. [Accessed 23 April 2015].

- [11] C. S. Corp. (2012). "Big data universe beginning to explode" [Online]. Available: http://www.csc.com/insights/flxwd/78931-big_data_universe_beginning_to_explode. [Accessed 23 April 2015].
- [12] B. Walker. (2015, Apr. 5). "Very day big data statistics" [Online]. Available: <http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/>. [Accessed 18 April 2016].
- [13] "Amazon Web Services" [Online]. Available: <http://aws.amazon.com/about-aws/>. [Accessed 22 April 2015].
- [14] A. Sharma. (2011, Aug. 19). "Ashish Sharma's tech blog" [Online]. Available: <http://www.ashishsharma.me/2011/08/external-merge-sort.html>. [Accessed 11 Sep. 2015].
- [15] Oracle. (2015). "Oracle Java documentation" [Online]. Available: <https://docs.oracle.com/javase/tutorial/collections/algorithms/#sorting>. [Accessed 11 Sep. 2015].
- [16] T. White, *Hadoop The Definitive Guide*, Sebastopol, CA: O'Reilly Media, Inc, 2015.
- [17] C. Nyberg, "Sort benchmark home page" [Online]. Available: <http://sortbenchmark.org/>. [Accessed 26 Aug. 2016].
- [18] V. Prajapati. (2015, Apr. 20). "How to install Apache Hadoop 2.6.0 in Ubuntu (Multi node/Cluster setup)" [Online]. Available: <http://pingax.com/install-apache-hadoop-ubuntu-cluster-setup/>. [Accessed 10 Aug. 2015].
- [19] Apache. (2014, Nov. 13). "MapReduce Tutorial about Hadoop 2.6.0 (Apache)," Apache [Online]. Available: http://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Source_Code. [Accessed 17 Nov. 2015].
- [20] "Ubuntu handbook," (2013, Oct. 5) [Online]. Available: <http://ubuntuhandbook.org/index.php/how-to-install-ubuntu/>. [Accessed 24 Nov. 2015].
- [21] Siva. (2015, Apr. 28). "Hadoop performance tuning" [Online]. Available: <http://hadooptutorial.info/hadoop-performance-tuning/>. [Accessed 15 Sep. 2015].

- [22] E. Sammer, *Hadoop Operations*, Beijing, Cambridge, Farnham, Koln, Sebastopol, Tokyo: O'Reilly Media, Inc., 2012.
- [23] D. Pollak, "Introduction," in *Beginning Scala*, New York: Octal Publishing, Inc., 2009, p. 23.
- [24] Apache. (2014, Nov. 13). "Hadoop cluster setup (Apache)," Apache [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html#Configuring_Environment_of_Hadoop_Daemons. [Accessed 22 April 2015].

Appendix A: Hadoop 2.6.0 Installation on Ubuntu 14.04

1. Purpose of This Appendix

This document intends to demonstrate the way to install Hadoop 2.6.0 on Ubuntu 14.04 as part of testing of the performance of Hadoop DFS.

2. Hadoop 2.6.0 Overview

Hadoop 2.6.0 is an updated version of Hadoop, but not the newest one which is Hadoop 2.7.1. There are mainly two reasons for me to choose this version: one is that this version is more powerful in its executive functions than the previous ones, and it provides torrents of available previous experiences; the other is that compared to its newest version, I believe this one has more advantage in supplying me with more solution options in fixing those unexpected errors since it has been in existence for a longer time and was used by more people.

3. Design of Lab Experiment

This installation was based on the installation guide [18].

(1) Install the relevant software which are required for Hadoop:

- I. Install Operating System, Ubuntu 14.04.2
- II. Install Java Development Kit, JDK 1.7.0_79
- III. Install Secure Shell, SSH-2.0-OpenSSH_5.9

(2) Set up before installing Hadoop:

- I. Adding a user group, adding a user, and authorizing the user

II. Disable IPv6, because Apache Hadoop is not supported on IPv6 networks. It has only been tested and developed on IPv4 stacks. Hadoop needs IPv4 to work, and only IPv4 clients can talk to the cluster.

(3) Install Hadoop 2.6.0

(4) Setting up configuration files

I. ~/.bashrc

II. /usr/local/hadoop/etc/hadoop/hadoop-env.sh

III. /usr/local/hadoop/etc/hadoop/core-site.xml

IV. /usr/local/hadoop/etc/hadoop/mapred-site.xml

V. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

VI. /usr/local/hadoop/etc/hadoop/yarn-site.xml

(5) Format the new Hadoop FS and Start Hadoop on single machine

(6) Networking building and Start Hadoop on the cluster

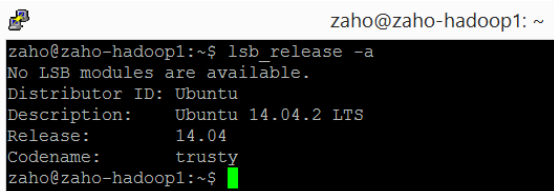
(7) Testing Word Count MapReduce program

4. Details and Result of Lab Experiment

(1) Install the relevant software items

I. Installing Ubuntu 14.04.2

Check: lsb_release -a



```

zaho@zaho-hadoop1: ~
zaho@zaho-hadoop1:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.2 LTS
Release:        14.04
Codename:       trusty
zaho@zaho-hadoop1:~$

```

II. Installing JDK 1.7.0_79

Command: sudo apt-get install openjdk-7-jdk

Check: java -version

```
zaho@zaho-hadoop1:~$ java -version
java version "1.7.0_79"
OpenJDK Runtime Environment (IcedTea 2.5.6) (7u79-2.5.6-0ubuntu1.14.04.1)
OpenJDK 64-Bit Server VM (build 24.79-b02, mixed mode)
zaho@zaho-hadoop1:~$
```

III. Installing ssh

Command: sudo apt-get install ssh

Check: which ssh

```
which sshd
zaho@zaho-hadoop1:~$ which ssh
/usr/bin/ssh
zaho@zaho-hadoop1:~$ which sshd
/usr/sbin/sshd
zaho@zaho-hadoop1:~$
```

(2) Configuration before installing Hadoop

I. Add a user group, add a user, and authorize the user

Command: sudo addgroup hadoop

```
sudo adduser --ingroup hadoop hduser
sudo adduser hduser sudo //authorized hduser sudo right
su hduser
ssh-keygen -t rsa -P ""
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

/* the last two lines mean to add the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password */

II. Disable ipv6

Add code in file: /etc/sysctl.conf

```
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Check: sudo sysctl -p

Or: cat /proc/sys/net/ipv6/conf/all/disable_ipv6

```
zaho@zaho-hadoop1:~$ sudo sysctl -p
[sudo] password for zaho:
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
zaho@zaho-hadoop1:~$
```

(3) Install Hadoop 2.6.0

Download Hadoop 2.6.0 in /usr/local/hadoop, command:

```
sudo su
wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
```

Install Hadoop, command:

```
tar xvzf hadoop-2.6.0.tar.gz
sudo mv * /usr/local/hadoop //copy * from hadoop-2.6.0 folder
sudo chown -R hduser:hadoop /usr/local/hadoop
```

(4) Setup configuration files

Note: This step should be done under hduser.

I. ~/.bashrc, add code at the end of the file, command: vi ~/.bashrc

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
#HADOOP VARIABLES END
```

```
source ~/.bashrc
javac -version
which javac
readlink -f /usr/bin/javac /usr/lib/jvm/java-7-openjdk-amd64/bin/javac
```

II. /usr/local/hadoop/etc/hadoop/hadoop-env.sh

Add code at the end of the file

```
#JAVA_HOME SETTING
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_PREFIX=/usr/local/hadoop
```

III. /usr/local/hadoop/etc/hadoop/core-site.xml

Command: sudo mkdir -p /app/hadoop/tmp

```
sudo chown hduser:hadoop /app/hadoop/tmp
```

add code in the file

```

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://zaho-hadoop1:54310</value>
    <description>The name of the default file system. A URI whose
    scheme and authority determine the FileSystem implementation. The
    uri's scheme determines the config property (fs.SCHEME.impl) naming
    the FileSystem implementation class. The uri's authority is used to
    determine the host, port tec. for a filesystem. </description>
  </property>
</configuration>

```

IV. /usr/local/hadoop/etc/hadoop/mapred-site.xml

There exists a file named mapred-site.xml.template, and create a new one named mapred-site.xml. Add code in the file.

```

<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>zaho-hadoop1:54311</value>
    <description>The host and port that the MapReduce job tracker runs
    at. If "local", then jobs are run in-process as a single map and
    reduce task.</description>
  </property>
</configuration>

```

V. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

Command: `sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode`

```

sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
sudo chown -R hduser:hadoop /usr/local/hadoop_store

```

Add code in the file

```

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
    <description>Default block replication.
    The actual number of replications can be specified when the file is created.
    The default is used if replication is not specified in create time.
  </description>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>

```

VI. /usr/local/hadoop/etc/hadoop/yarn-site.xml

And code in the file

```

<configuration>
<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>

```

(5) Format the new Hadoop FS and Start Hadoop on single machine

I. Under /usr/local/hadoop_store/hdfs/namenode, execute command:

```
Hadoop namenode -format
```

Note: This command should be executed once before we start using Hadoop. If this command is executed again after Hadoop has been used, it'll destroy all the data on the Hadoop file system.

II. Under /usr/local/hadoop/sbin, execute command:

```
start-all.sh
```

```
jps //check if it really starts
```

```
stop-all.sh //stop all services
```

(6) Networking building and Start Hadoop on the cluster

I. Modify the file /etc/hosts, command: sudo vi /etc/hosts

Note: use command ifconfig to find the ip for each machine first.

```

hduser@zaho-hadoop1: /home/zaho
#27.0.0.1    localhost
#127.0.0.1  zaho-hadoop1
10.59.7.42  zaho-hadoop1 #master
10.59.7.43  zaho-hadoop2
10.59.7.44  zaho-hadoop3
10.59.7.45  zaho-hadoop4
10.59.7.46  zaho-hadoop5
10.59.7.47  zaho-hadoop6
10.59.7.48  zaho-hadoop7
10.59.7.49  zaho-hadoop8
# The following lines are desirable for IPv6 capable hosts
::1        localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
~
~
~
~
~
~
~
~
~
~/etc/hosts" 14L, 422C

```

II. Copy the file hosts to each machine

Note: remove hosts first, command: `rm hosts`

Command: `sudo scp hduser@zaho-hadoop1: /etc/hosts /etc/`

III. Under master machine (zaho-hadoop1), execute command for all other machines:

```
ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@zaho-hadoop2
```

(my slaves: zaho-hadoop2, zaho-hadoop3, zaho-hadoop4, zaho-hadoop5, zaho-hadoop6, zaho-hadoop7, zaho-hadoop8)

Note: After this command, you can ssh to each machine without password any more.

IV. Create/modify file, masters and slaves under /usr/local/hadoop/etc/hadoop

```
hduser@zaho-hadoop1:/usr/local/hadoop/etc/hadoop$ vi masters
zaho-hadoop1
~
hduser@zaho-hadoop1:/usr/local/hadoop/etc/hadoop$ vi slaves
zaho-hadoop1
zaho-hadoop2
zaho-hadoop3
zaho-hadoop4
zaho-hadoop5
zaho-hadoop6
zaho-hadoop7
zaho-hadoop8
~
```

Note: zaho-hadoop1 machine will be a master and a slave.

Then copy these two files to other machines.

V. Reset core-site.xml and mapred-site.xml for multiple nodes

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://zaho-hadoop1:54310</value>
    <description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used to
determine the host, port tec. for a filesystem. </description>
  </property>
</configuration>
```

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>zaho-hadoop1:54311</value>
    <description>The host and port that the MapReduce job tracker runs
    at. If "local", then jobs are run in-process as a single map and
    reduce task.</description>
  </property>
</configuration>
```

Then copy these two files to each machine

VI. Start Hadoop as a cluster.

start-all.sh

jps //check if it really starts

```
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
15/08/24 11:12:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [zaho-hadoop1]
zaho-hadoop1: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-zaho-hadoop1.out
zaho-hadoop3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop3.out
zaho-hadoop7: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop7.out
zaho-hadoop2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop2.out
zaho-hadoop5: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop5.out
zaho-hadoop6: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop6.out
zaho-hadoop8: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop8.out
zaho-hadoop1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop1.out
zaho-hadoop4: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-zaho-hadoop4.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-zaho-hadoop1.out
15/08/24 11:12:26 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-zaho-hadoop1.out
zaho-hadoop1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop1.out
zaho-hadoop7: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop7.out
zaho-hadoop3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop3.out
zaho-hadoop5: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop5.out
zaho-hadoop2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop2.out
zaho-hadoop8: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop8.out
zaho-hadoop6: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop6.out
zaho-hadoop4: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-zaho-hadoop4.out
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ jps
20758 SecondaryNameNode
21366 Jps
20907 ResourceManager
20390 NameNode
20541 DataNode
21054 NodeManager
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ ssh zaho-hadoop2
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-54-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Mon Aug 24 11:12:19 CDT 2015
```

```
System information as of Mon Aug 24 11:12:19 CDT 2015

System load: 0.0          Processes: 89
Usage of /: 31.9% of 15.62GB  Users logged in: 0
Memory usage: 22%        IP address for eth0: 10.59.7.43
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

52 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:00:21 2015 from zaho-hadoop1
hduser@zaho-hadoop2:~$ jps
2127 DataNode
2454 Jps
2272 NodeManager
hduser@zaho-hadoop2:~$ ssh zaho-hadoop3
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:12:22 CDT 2015

System load: 0.14         Processes: 89
Usage of /: 30.9% of 15.62GB  Users logged in: 0
Memory usage: 23%        IP address for eth0: 10.59.7.44
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:00:45 2015 from zaho-hadoop2
hduser@zaho-hadoop3:~$ jps
32752 DataNode
590 Jps
439 NodeManager
hduser@zaho-hadoop3:~$ ssh zaho-hadoop4
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:15:03 CDT 2015
```

```

System information as of Mon Aug 24 11:15:03 CDT 2015

System load: 0.0          Processes: 91
Usage of /: 31.1% of 15.62GB  Users logged in: 1
Memory usage: 23%        IP address for eth0: 10.59.7.45
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:04:04 2015 from zaho-hadoop3
hduser@zaho-hadoop4:~$ jps
29959 DataNode
30104 NodeManager
30289 Jps
hduser@zaho-hadoop4:~$ ssh zaho-hadoop5
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:10:09 CDT 2015

System load: 0.21       Processes: 89
Usage of /: 30.9% of 15.62GB  Users logged in: 0
Memory usage: 21%      IP address for eth0: 10.59.7.46
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Fri Jul 17 22:42:36 2015 from zaho-hadoop1
hduser@zaho-hadoop5:~$ jps
28543 NodeManager
28725 Jps
28398 DataNode
hduser@zaho-hadoop5:~$ ssh zaho-hadoop6
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:10:09 CDT 2015

```



```

System information as of Mon Aug 24 11:10:09 CDT 2015

System load: 0.0          Processes:          90
Usage of /: 30.9% of 15.62GB Users logged in: 0
Memory usage: 21%        IP address for eth0: 10.59.7.47
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Fri Jul 17 22:43:03 2015 from zaho-hadoop1
hduser@zaho-hadoop6:~$ jps
29390 DataNode
29717 Jps
29535 NodeManager
hduser@zaho-hadoop6:~$ ssh zaho-hadoop7
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:14:40 CDT 2015

System load: 0.07         Processes:          93
Usage of /: 31.1% of 15.62GB Users logged in: 1
Memory usage: 23%        IP address for eth0: 10.59.7.48
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Fri Jul 17 22:46:44 2015 from zaho-hadoop1
hduser@zaho-hadoop7:~$ jps
29711 DataNode
29856 NodeManager
30037 Jps
hduser@zaho-hadoop7:~$ ssh zaho-hadoop8
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:11:57 CDT 2015
System information as of Mon Aug 24 11:11:57 CDT 2015

System load: 0.21         Processes:          91
Usage of /: 31.1% of 15.62GB Users logged in: 1
Memory usage: 22%        IP address for eth0: 10.59.7.49
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Fri Jul 17 22:45:34 2015 from zaho-hadoop1
hduser@zaho-hadoop8:~$ jps
30802 DataNode
31124 Jps
30947 NodeManager
hduser@zaho-hadoop8:~$ █

```

```

hduser@zaho-hadoop8:~$ exit
logout
Connection to zaho-hadoop8 closed.
hduser@zaho-hadoop7:~$ exit
logout
Connection to zaho-hadoop7 closed.
hduser@zaho-hadoop6:~$ exit
logout
Connection to zaho-hadoop6 closed.
hduser@zaho-hadoop5:~$ exit
logout
Connection to zaho-hadoop5 closed.
hduser@zaho-hadoop4:~$ exit
logout
Connection to zaho-hadoop4 closed.
hduser@zaho-hadoop3:~$ exit
logout
Connection to zaho-hadoop3 closed.
hduser@zaho-hadoop2:~$ exit
logout
Connection to zaho-hadoop2 closed.
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ █

```

stop-all.sh //stop all services

```

hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ stop-all.sh
This script is deprecated. Instead use stop-dfs.sh and stop-yarn.sh
15/08/24 11:36:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Stopping namenodes on [zaho-hadoop1]
zaho-hadoop1: stopping namenode
zaho-hadoop2: stopping datanode
zaho-hadoop3: stopping datanode
zaho-hadoop7: stopping datanode
zaho-hadoop6: stopping datanode
zaho-hadoop1: stopping datanode
zaho-hadoop5: stopping datanode
zaho-hadoop8: stopping datanode
zaho-hadoop4: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
15/08/24 11:36:39 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
stopping yarn daemons
stopping resourcemanager
zaho-hadoop1: stopping nodemanager
zaho-hadoop7: stopping nodemanager
zaho-hadoop2: stopping nodemanager
zaho-hadoop8: stopping nodemanager
zaho-hadoop6: stopping nodemanager
zaho-hadoop5: stopping nodemanager
zaho-hadoop4: stopping nodemanager
zaho-hadoop3: stopping nodemanager
zaho-hadoop7: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop2: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop8: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop6: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop5: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop4: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
zaho-hadoop3: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
no proxyserver to stop
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ jps
23829 Jps
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ ssh zaho-hadoop2
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Aug 24 11:36:36 CDT 2015

```

```
System information as of Mon Aug 24 11:36:36 CDT 2015

System load: 0.22          Processes: 90
Usage of /: 31.9% of 15.62GB Users logged in: 0
Memory usage: 24%        IP address for eth0: 10.59.7.43
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

52 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:34:15 2015 from zaho-hadoop1
hduser@zaho-hadoop2:~$ jps
3240 Jps
hduser@zaho-hadoop2:~$ ssh zaho-hadoop3
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:36:38 CDT 2015

System load: 0.09          Processes: 91
Usage of /: 30.9% of 15.62GB Users logged in: 0
Memory usage: 25%        IP address for eth0: 10.59.7.44
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:14:11 2015 from zaho-hadoop2
hduser@zaho-hadoop3:~$ jps
1278 Jps
hduser@zaho-hadoop3:~$ ssh zaho-hadoop4
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:39:20 CDT 2015
```

```
System information as of Mon Aug 24 11:39:20 CDT 2015

System load: 0.04          Processes:          90
Usage of /: 31.1% of 15.62GB  Users logged in: 1
Memory usage: 25%         IP address for eth0: 10.59.7.45
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:17:06 2015 from zaho-hadoop3
hduser@zaho-hadoop4:~$ jps
30991 Jps
hduser@zaho-hadoop4:~$ ssh zaho-hadoop5
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Aug 24 11:34:25 CDT 2015

System load: 0.21          Processes:          89
Usage of /: 30.9% of 15.62GB  Users logged in: 0
Memory usage: 22%         IP address for eth0: 10.59.7.46
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:12:31 2015 from zaho-hadoop4
hduser@zaho-hadoop5:~$ jps
29430 Jps
hduser@zaho-hadoop5:~$ ssh zaho-hadoop6
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Aug 24 11:34:25 CDT 2015
```

```

System information as of Mon Aug 24 11:34:25 CDT 2015

System load: 0.18          Processes: 91
Usage of /: 30.9% of 15.62GB  Users logged in: 0
Memory usage: 23%         IP address for eth0: 10.59.7.47
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:12:52 2015 from zaho-hadoop5
hduser@zaho-hadoop6:~$ jps
30422 Jps
hduser@zaho-hadoop6:~$ ssh zaho-hadoop7
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:38:57 CDT 2015

System load: 0.07          Processes: 95
Usage of /: 31.1% of 15.62GB  Users logged in: 1
Memory usage: 24%         IP address for eth0: 10.59.7.48
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:17:56 2015 from zaho-hadoop6
hduser@zaho-hadoop7:~$ jps
30738 Jps
hduser@zaho-hadoop7:~$ ssh zaho-hadoop8
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation: https://help.ubuntu.com/

System information as of Mon Aug 24 11:36:13 CDT 2015

```

```

System information as of Mon Aug 24 11:36:13 CDT 2015

System load: 0.17          Processes: 92
Usage of /: 31.1% of 15.62GB  Users logged in: 1
Memory usage: 23%         IP address for eth0: 10.59.7.49
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

102 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Aug 24 11:15:31 2015 from zaho-hadoop7
hduser@zaho-hadoop8:~$ jps
31826 Jps
hduser@zaho-hadoop8:~$ exit
logout
Connection to zaho-hadoop8 closed.
hduser@zaho-hadoop7:~$ exit
logout
Connection to zaho-hadoop7 closed.
hduser@zaho-hadoop6:~$ exit
logout
Connection to zaho-hadoop6 closed.
hduser@zaho-hadoop5:~$ exit
logout
Connection to zaho-hadoop5 closed.
hduser@zaho-hadoop4:~$ exit
logout
Connection to zaho-hadoop4 closed.
hduser@zaho-hadoop3:~$ exit
logout
Connection to zaho-hadoop3 closed.
hduser@zaho-hadoop2:~$ exit
logout
Connection to zaho-hadoop2 closed.
hduser@zaho-hadoop1:/usr/local/hadoop/sbin$ █

```

(7) Testing Word Count MapReduce program [19]

I. Environment variables are set

```
export JAVA_HOME=/usr/java/default
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
```

II. Compiling the program and creating jar file

```
/usr/local/hadoop/bin/hadoop com.sun.tools.javac.Main WordCount.java
jar cf hs.jar WordCount*.class
```

```
hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hadoop com.sun.tools.java
c.Main WordCount.java
hduser@zaho-hadoop1:~/wordCount$ jar cf hs.jar WordCount*.class
hduser@zaho-hadoop1:~/wordCount$ ls
CreateData.class  hs.jar          WordCount$IntSumReducer.class
CreateData.java   wc.jar          WordCount.java
dataLog           WordCount.class WordCount$TokenizerMapper.class
hduser@zaho-hadoop1:~/wordCount$
```

III. Creating input and output folder in HDFS and data files as input

```
/usr/local/hadoop/bin/hdfs dfs -mkdir /user/zhao/input
/usr/local/hadoop/bin/hdfs dfs -mkdir /user/zhao/output
/usr/local/hadoop/bin/hdfs dfs -put input/* /user/zhao/input
```

```
hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hdfs dfs -mkdir /user/zha
o/input
15/11/16 10:10:35 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hdfs dfs -mkdir /user/zha
o/output
15/11/16 10:10:44 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hdfs dfs -put input/* /us
er/zhao/input
15/11/16 10:16:31 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hduser@zaho-hadoop1:~/wordCount$
```

IV. Running the program

```
/usr/local/hadoop/bin/hadoop jar wc.jar WordCount /user/zhao/input /user/zhao/output/wc
```

```
hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hadoop jar wc.jar WordCount /user/zhao/input /user/zhao/output/wc
15/11/16 10:18:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/11/16 10:18:31 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
15/11/16 10:18:31 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
15/11/16 10:18:31 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
15/11/16 10:18:31 INFO input.FileInputFormat: Total input paths to process : 2
15/11/16 10:18:31 INFO mapreduce.JobSubmitter: number of splits:2
15/11/16 10:18:32 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1635850916_0001
15/11/16 10:18:32 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
15/11/16 10:18:32 INFO mapreduce.Job: Running job: job_local1635850916_0001
15/11/16 10:18:32 INFO mapred.LocalJobRunner: OutputCommitter set in config null
15/11/16 10:18:32 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
15/11/16 10:18:33 INFO mapred.LocalJobRunner: Waiting for map tasks
15/11/16 10:18:33 INFO mapred.LocalJobRunner: Starting task: attempt_local1635850916_0001_m_000000_0
15/11/16 10:18:33 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
15/11/16 10:18:33 INFO mapred.MapTask: Processing split: hdfs://zaho-hadoop1:54310/user/zhao/input/file1:0484
15/11/16 10:18:33 INFO mapred.MapTask: (EQUATOR) 0 kv1 26214396(104857584)
15/11/16 10:18:33 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
15/11/16 10:18:33 INFO mapred.MapTask: soft limit at 8386080
15/11/16 10:18:33 INFO mapred.MapTask: bufstart = 0; bufend = 104857600
15/11/16 10:18:33 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
15/11/16 10:18:33 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
15/11/16 10:18:33 INFO mapred.LocalJobRunner:
15/11/16 10:18:33 INFO mapred.MapTask: Starting flush of map output
15/11/16 10:18:33 INFO mapred.MapTask: Spilling map output
15/11/16 10:18:33 INFO mapred.MapTask: bufstart = 0; bufend = 160; bufvoid = 104857600
15/11/16 10:18:33 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214324(104857296); length = 73/6553600
15/11/16 10:18:33 INFO mapred.MapTask: Finished spill 0
```

```

15/11/16 10:18:33 INFO mapreduce.Job: map 100% reduce 100%
15/11/16 10:18:33 INFO mapreduce.Job: Job job_local1635850916_0001 completed successfully
15/11/16 10:18:33 INFO mapreduce.Job: Counters: 38
  File System Counters
    FILE: Number of bytes read=11436
    FILE: Number of bytes written=765573
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=388
    HDFS: Number of bytes written=176
    HDFS: Number of read operations=25
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=5
  Map-Reduce Framework
    Map input records=6
    Map output records=33
    Map output bytes=283
    Map output materialized bytes=319
    Input split bytes=224
    Combine input records=33
    Combine output records=28
    Reduce input groups=24
    Reduce shuffle bytes=319
    Reduce input records=28
    Reduce output records=24
    Spilled Records=56
    Shuffled Maps =2
    Failed Shuffles=0
    Merged Map outputs=2
    GC time elapsed (ms)=9
    CPU time spent (ms)=0
    Physical memory (bytes) snapshot=0
    Virtual memory (bytes) snapshot=0
    Total committed heap usage (bytes)=936902656
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=152
  File Output Format Counters
    Bytes Written=176
hduser@zaho-hadoop1:~/wordCount$ █

```

V. Output file of result

```
/usr/local/hadoop/bin/hdfs dfs -cat /user/zhao/output/wc/part-r-00000
```

```

hduser@zaho-hadoop1:~/wordCount$ /usr/local/hadoop/bin/hdfs dfs -cat /user/zhao/outp
ut/wc/part-r-00000
15/11/16 10:24:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library f
or your platform... using builtin-java classes where applicable
Can      1
GA       1
I        4
a        4
am       2
and      3
before.  1
consultant      1
for          1
give         1
graduate      1
job          1
lab          1
like         1
looking      1
me           1
music.       1
now.         1
offer?       1
right        1
sports       1
student.     1
was          1
you          1
hduser@zaho-hadoop1:~/wordCount$ █

```

5. Lessons Learned

No.	Issues faced	Solutions
1	The zaho-hadoop3 machine cannot start up when starting Hadoop cluster	Re-execute the relevant commands under the user hduser.
2	Nodemanager does not show up when jps. After restarting the cluster, nodemanager shows up, but after a while, cannot see it when jps entered	The fact proves that nodemanager is running, but just not shows up when excuting jps, which does not impact the job execution result.

6. Recommendations

Before doing every command, we have to first confirm under which user the command should be executed: hduser or root or sudo su, because the result of the execution will affect the authorization for the related files or folders.

7. Exit Criteria

- a) All machines of Hadoop cluster can start and stop service correctly - **Yes**
- b) Word Count MapReduce program can be executed successfully – **Yes**

8. Conclusion

As the Exit Criteria I set was met and satisfied as mentioned in Section 7, the Hadoop cluster is successfully installed. It can be used for the next lab experiments.

Appendix B: Centralized File System Performance Test

1. Purpose

This appendix measures the performance of the centralized file system on (Ext4) [20] a single test machine. The data includes read time, write time, sorting time, and total time, which will be compared to the result of Hadoop DFS cluster execution.

It does this through the use of the sorting programs in Section 2, and running the test cases identified in Section 3. The result is presented in Section 4 and discussed in Section 5, 6, and 7.

2. Programs for testing

(1) Creating data program, Internal sort program and External sort program are all discussed in the main part of this paper.

(2) Result checking program

I. Description

The purpose of this program is to check if the output file is sorted correctly. When running this program, we should add the file name to be tested in the command line, and then the test result will be printed out on the screen.

II. Code. As shown in Source Code 4: CheckResult.java.

3. Designing test cases and result data explanation are talked about in main part of this paper.

4. Lessons Learned

No.	Issues faced	Solutions
1	When file size is larger than free memory size, how to sort the data file?	External sort algorithm can solve this problem
2	If the free disk space is 30 GB, only 10 GB data file can be tested because the temporary medium files are actually a copy of original file and the output file is another copy.	Modified the code: after having generated all medium files, the program will delete the original file so as to save more disk space to store output file. After modification, 15 GB data file can be tested when there is 30 GB of free disk space.
3	If the data file is big enough, the memory will be eaten up.	The buffer size can be set smaller when in merge phase of the External sort algorithm
4	When dealing with more than one file, recalling the splitting data file function will need too much time to recreate objects in memory.	Use another algorithm: first, concatenating the data files, and then running the external sort program.

5. Recommendations

When running the programs to test data files, we should always pay attention to the boundary values, such as the free memory size, free disk space size, the range of int variable type, and so on. In these cases, we can use different algorithms or different programs or different types, otherwise you will get error or wrong result.

6. Conclusion

The programs and algorithms worked correctly, and the necessary data was collected for the research paper. The code can be examined by any other individual who intends to check my result data.

Appendix C: Hadoop DFS Performance Test

1. Purpose

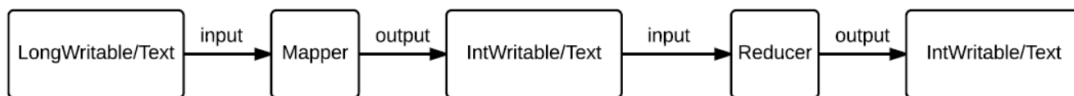
This appendix will collect the result data of a DFS from a Hadoop cluster test. The data includes read time, write time, sorting time, and total time. These will be used to compare with the result of a single machine test and adjust relevant parameters for better performance.

2. Optimizing for MapReduce Programs and Hadoop Setting

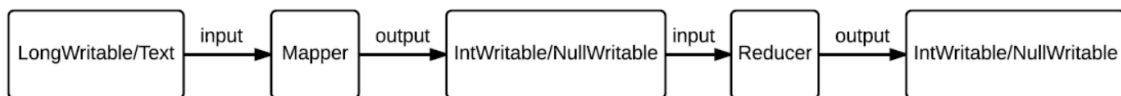
2.1 Optimizing MapReduce program

I. Description

In the old version, the output type of mapper is `IntWritable/Text`, while in the new version, the output type of mapper is `IntWritable/NullWritable`. Also, in the old version input and output type of reducer is `IntWritable/Text`, while in the final version, those of reducer are `IntWritable/NullWritable` as shown in Figure 1. The change of the type significantly improved the performance.



Old version flow chart



Final version flow chart

Note: `LongWritable`, `IntWritable`, `Text` and `NullWritable` are the different types in the Hadoop. `NullWritable` is similar to `null` in java, but `NullWritable` is a type, not a value.

II. Code

- i) Old version. As shown in Source Code 5: HadoopSort.java (the version before optimizing).
- ii) Final version. As shown in Source Code 6: HadoopSort.java (the version after optimizing).

III. Result of testing (4 GB data file, default Hadoop Setting) (Total Time minutes)

Code Version	Test 1	Test 2	Test 3	Average	Δ
Old	47.75	47.07	49.28	48.03	1.13
Final	42.52	42.17	41.23	41.97	0.67
Δ	5.23	4.9	8.05	6.06	

Note 1: Average = (Test1 + Test2 + Test3)/3.

Note 2: $\Delta = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$. This formula is for Sample case, not for Population case.

From the comparison of average total time between two code versions, the final version apparently runs faster than the old one by about 6.06 minutes. Also, the final version is more stable than the old one because the standard deviation of the final is lower than the old one about 0.46 from the data of Δ . In conclusion, the final version is much better than the old.

2.2 Optimizing Hadoop setting

I. Description

Hadoop provides a set of parameters on CPU, memory, disk, and network for performance tuning. Most Hadoop tasks are not CPU-bound; what we usually look into is how to optimize usage of memory and disk spills.

II. Results of parameter tunings (Using 4G file size)

Hadoop/FileSize(4G)												
Tuning Case	Hadoop Configuration						Total Time(min)					Notes
Parameter	TotalAmount OfBuffer	threshold at which spill is triggered	MEM sizes available to the JVM	NumOf Reducer	master+ slaves	Data Copies	Low	Medium	High	Average	Δ	
name	mapreduce.task.io.sort.mb	mapreduce.map.sort.spill.percent	mapred.child.java.opts	mapreduce.job.reduces		dfs.replication						
default Value	100	0.8	-Xmx200m	1		3						
1	100	0.8	-Xmx200m	1	1+7	3	41.78	43.57	45.33	43.56	1.78	
2	100	0.8	-Xmx200m	1	1+8	3	43.83	44.43	46.7	44.99	1.51	
3	100	0.8	-Xmx200m	1	1+7	1	42.47	43.82	44.87	43.72	1.20	
4	100	0.8	-Xmx200m	1	1+7	2	42.5	44.12	44.63	43.75	1.11	
5	100	0.8	-Xms1024M Xmx2048M	1	1+7	3	42.85	42.88	46.05	43.93	1.84	
6	200	0.8	-Xms1024M Xmx2048M	1	1+7	3	41.47	41.83	43.01	42.10	0.81	
7	400	0.8	-Xms1024M Xmx2048M	1	1+7	3					outOfMemError	
8	300	0.8	-Xms1024M Xmx2048M	1	1+7	3	41.82	45.25	45.68	44.25	2.12	
9	300	0.8	-Xms1024M Xmx2048M	2	1+7	3	41.78	42.57	44.72	43.02	1.52	
10	300	0.8	-Xms2048M Xmx2048M	2	1+7	3	41.52	41.78	45.68	42.99	2.33	
11	300	0.8	-Xms1024M Xmx1024M	2	1+7	3	39.82	41.27	44.98	42.02	2.66	
12	300	0.9	-Xms1024M Xmx1024M	2	1+7	3	40.29	41.58	42.08	41.32	0.92	
13	286	1	-Xms1024M Xmx1024M	2	1+7	3	41.63	42.35	43.85	42.61	1.13	
14	300	1	-Xms1024M Xmx1024M	2	1+7	3	35.35	35.45	36.13	35.64	0.42	
15	300	1	-Xms1024M Xmx1024M	2	1+7	3	41.20	42.20	42.5	41.97	0.68	mapreduce.job.jvm.numtasks = -1

Note 1: I modified the configuration files (*masters and slaves*) to check the influence on performance for different values of some parameters. The result of tuning cases 1 and 2 shows that one master and seven slaves (1+7) is better than one master and eight slaves (1+8).

Note 2: The parameter of *data copies* was changed in cases 1, 3 and 4. The result shows that the number of replications (data copies) does not affect the performance a lot, and we do not choose one copy because one copy lowers the security of the Hadoop.

Note 3: The result of case 11 shows a 5 minutes' difference between High and Low for Total Time. This indicates that this configuration of Hadoop system has high variability.

Note 4: I modified the parameter: number of reducers in cases 8 and 9. The result shows that setting up 2 reducers is little bit better than 1 reducer in performance.

Note 5: The parameter of "Total Amount of Buffer" was changed in cases 5, 6 and 8. The result proved this parameter is the most sensitive for improving performance.

Note 6: The parameter of "Number of Mapper" was not chosen because the value of this parameter is only a suggestion, and the number of split pieces (Hadoop will split the input data file into smaller pieces based on the block size, default value is 128 MB) decides the real number of mappers.

Note 7: Total Amount of Buffer can be calculated by a formula: $(16 + R) * N/1048576$, of which $R = \text{Map output bytes}/\text{Map output records}$, and $N = \text{Map output records}/\# \text{ of Map tasks}$. As Map output is being sorted, 16 bytes of metadata are added immediately before each key-value pair. These 16 bytes include 12 bytes for the key-value offset and 4 bytes for the indirect-sort index. 1048576 is 1 MB, a unit of Buffer size in memory. The value of the Total Amount of Buffer calculated by this formula will lead to the least number of spills for each Map tasks. [21]

Note 8: $\Delta = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$. This formula is to calculate the standard deviation for Sample case.

Note 9: I modified the parameter, “*mapreduce.job.jvm.numtasks*” to cause JVM to be reused instead of being re-created. I thought this would save time because JVM will stay alive without opening and closing the JVM many times. However, the resulting execution time of tuning case 15 in comparison to case 14 was worse.

3. Designing test cases and result data are talked about in the main part of the paper

The approach to calculate read time, write time, and sort time is the following. Firstly, run TestDFSIO to achieve read throughput and write throughput. Each test was executed three times, and their arithmetic average value will be used; secondly, calculate the read time by the formula “File Size/read throughput/60”, similarly for write time; finally, use the formula “total time – read time – write time” to obtain sort time. Consider Test number 1 as an example:

Step 1, to execute the TestDFSIO build-in jar to obtain the read throughput and write throughput.

```
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ yarn jar hadoop-mapreduce-client-jobclient-2.6.0.jar TestDFSIO -read -nrFiles 1 -fileSize 1156
16/02/20 08:59:59 INFO fs.TestDFSIO: TestDFSIO.1.7
16/02/20 08:59:59 INFO fs.TestDFSIO: nrFiles = 1
16/02/20 08:59:59 INFO fs.TestDFSIO: nrBytes (MB) = 1156.0
16/02/15 14:32:48 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
16/02/15 14:32:48 INFO fs.TestDFSIO: Date & time: Mon Feb 15 14:32:48
CST 2016
16/02/15 14:32:48 INFO fs.TestDFSIO: Number of files: 1
16/02/15 14:32:48 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:32:48 INFO fs.TestDFSIO: Throughput mb/sec: 376.0572543916721
16/02/15 14:32:48 INFO fs.TestDFSIO: Average IO rate mb/sec: 376.0572509765625
16/02/15 14:32:48 INFO fs.TestDFSIO: IO rate std deviation: 0.08054781612582725
16/02/15 14:32:48 INFO fs.TestDFSIO: Test exec time sec: 4.744
16/02/15 14:32:48 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █
```



```

16/02/15 14:33:27 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
16/02/15 14:33:27 INFO fs.TestDFSIO: Date & time: Mon Feb 15 14:33:27
CST 2016
16/02/15 14:33:27 INFO fs.TestDFSIO: Number of files: 1
16/02/15 14:33:27 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:33:27 INFO fs.TestDFSIO: Throughput mb/sec: 311.5063325249259
16/02/15 14:33:27 INFO fs.TestDFSIO: Average IO rate mb/sec: 311.50634765625
16/02/15 14:33:27 INFO fs.TestDFSIO: IO rate std deviation: 0.03879608858980715
16/02/15 14:33:27 INFO fs.TestDFSIO: Test exec time sec: 5.683
16/02/15 14:33:27 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █
16/02/15 14:34:50 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
16/02/15 14:34:50 INFO fs.TestDFSIO: Date & time: Mon Feb 15 14:34:50
CST 2016
16/02/15 14:34:50 INFO fs.TestDFSIO: Number of files: 1
16/02/15 14:34:50 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:34:50 INFO fs.TestDFSIO: Throughput mb/sec: 298.63084474296045
16/02/15 14:34:50 INFO fs.TestDFSIO: Average IO rate mb/sec: 298.6308898925781
16/02/15 14:34:50 INFO fs.TestDFSIO: IO rate std deviation: 0.13331553967838147
16/02/15 14:34:50 INFO fs.TestDFSIO: Test exec time sec: 5.766
16/02/15 14:34:50 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █

```

AVG Read Throughput = $(376.06 + 311.51 + 298.63) / 3 = 328.73$ (MB/sec)

```

hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ yarn jar hadoop-ma
produce-client-jobclient-2.6.0.jar TestDFSIO -write -nrFiles 1 -fileSize 1156
16/02/15 09:20:22 INFO fs.TestDFSIO: TestDFSIO.1.7
16/02/15 09:20:22 INFO fs.TestDFSIO: nrFiles = 1
16/02/15 09:20:22 INFO fs.TestDFSIO: nrBytes (MB) = 1156.0
16/02/15 14:36:56 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/02/15 14:36:56 INFO fs.TestDFSIO: Date & time: Mon Feb 15 14:36:56
CST 2016
16/02/15 14:36:56 INFO fs.TestDFSIO: Number of files: 1
16/02/15 14:36:56 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:36:56 INFO fs.TestDFSIO: Throughput mb/sec: 145.17141780735903
16/02/15 14:36:56 INFO fs.TestDFSIO: Average IO rate mb/sec: 145.17141723632812
16/02/15 14:36:56 INFO fs.TestDFSIO: IO rate std deviation: 0.01216671963276286
16/02/15 14:36:56 INFO fs.TestDFSIO: Test exec time sec: 9.726
16/02/15 14:36:56 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █
16/02/15 14:39:57 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/02/15 14:39:57 INFO fs.TestDFSIO: Date & time: Mon Feb 15 14:39:57
CST 2016
16/02/15 14:39:57 INFO fs.TestDFSIO: Number of files: 1
16/02/15 14:39:57 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:39:57 INFO fs.TestDFSIO: Throughput mb/sec: 143.06930693069307
16/02/15 14:39:57 INFO fs.TestDFSIO: Average IO rate mb/sec: 143.06930541992188
16/02/15 14:39:57 INFO fs.TestDFSIO: IO rate std deviation: 0.00430535866131636
84
16/02/15 14:39:57 INFO fs.TestDFSIO: Test exec time sec: 9.727
16/02/15 14:39:57 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █

```

```

16/02/15 14:39:13 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/02/15 14:39:13 INFO fs.TestDFSIO:                               Date & time: Mon Feb 15 14:39:13
CST 2016
16/02/15 14:39:13 INFO fs.TestDFSIO:                               Number of files: 1
16/02/15 14:39:13 INFO fs.TestDFSIO: Total MBytes processed: 1156.0
16/02/15 14:39:13 INFO fs.TestDFSIO:                               Throughput mb/sec: 147.5054229934924
16/02/15 14:39:13 INFO fs.TestDFSIO: Average IO rate mb/sec: 147.5054168701172
16/02/15 14:39:13 INFO fs.TestDFSIO: IO rate std deviation: 0.01870232745352778
16/02/15 14:39:13 INFO fs.TestDFSIO: Test exec time sec: 10.703
16/02/15 14:39:13 INFO fs.TestDFSIO:
hduser@zaho-hadoop1:/usr/local/hadoop/share/hadoop/mapreduce$ █

```

AVG Write Throughput = $(145.17 + 143.07 + 147.51) / 3 = 145.25$ (MB/sec)

Step 2, to calculate the read time and write time by using the formula

HDFS read bytes / read throughput / 60 = $1155.55/328.73/60 = 0.06$ minutes

HDFS write bytes / write throughput / 60 = $1155.55/145.25/60 = 0.13$ minutes

Step 3, to calculate the sort time by the formula (the value of total time comes from the average total time for Test 1 in Table 6: Hadoop cluster tested result data).

Total time – read time – write time = $10.05 - 0.06 - 0.13 = 9.86$ minutes

4. Lessons Learned

No.	Issues faced	Solutions
1	The Hadoop MapReduce program impacts the job performance. See Section 2.1.	To choose appropriate input and output key/value types will apparently improve the performance.
2	The Hadoop setting parameters influence the Hadoop job performance.	According to job features, changing the Hadoop setting parameters leads to better performance. See Section 2.2.
3	When restarting the Hadoop System after having modified the configuration, the Hadoop sometimes reports Error for stop-all.sh command.	To follow the steps: stop Hadoop system -> modify the configuration -> start Hadoop system. Note: it is not necessary to format the Namenode for just changing the configuration files, and formatting Namenode will delete everything from the HDFS.

5. Recommendations

When executing Hadoop jobs, we usually can optimize the MapReduce program and Hadoop parameters setting so that the Hadoop system's parallel computing advantages can be shown.

As for the MapReduce program optimizing, there are two options we usually need to be considered: one is the input and output key/value types; the other is the algorithms. As for the Hadoop parameter setting optimizing, there are four elements which will possibly become bottlenecks. Theoretically, they are CPU, Main Memory, Network Bandwidth, and Storage I/O. Practically, we focus on utilizing sufficiently Main Memory (tuning cases 6 – 11), and reducing I/O operations (tuning cases 12 – 14) and node communication, or decreasing the amount of data transferring between nodes and transferring between Memory and storage.

6. Conclusion

After optimizing of the Hadoop program and Hadoop parameter settings, the Hadoop jobs worked correctly and efficiently and the performance was improved.

Appendix D: Source Codes

➤ Source Code 1: CreateData.java

```

1  /**
2  * @Purpose: This class will generate file(s) by which
3  *           integers can be generated with specific
4  *           numbers and specific files
5  * @Author: Zhao Xie
6  * @Date: 8/27/2015
7  * @File: CreateData.java
8  */
9  import java.io.*;
10 import java.util.*;
11
12 public class CreateData{
13     private static final String FILENAME = "file"; //file name prefix
14     private static final int NUMOFFILES = 1; //number of files
15     private static final long NUMSPERFILE = 3500000000L;
16     private static final int MAXVALUE = 100000000;
17
18     public static void main(String[] args){
19         long writeStart = System.currentTimeMillis();
20         int count = 0;
21         try{
22             /* produce data files */
23             while(count<NUMOFFILES){
24                 String newFileName = FILENAME + count;
25                 File myFile = new File(newFileName);
26                 Random randomGenerator = new Random();
27                 FileWriter write = new FileWriter(myFile);
28                 write.write("");
29                 long n = NUMSPERFILE-1;
30                 for(long i=0;i<n;i++){
31                     write.append(randomGenerator.nextInt(MAXVALUE) + "\n");
32                 }
33                 //confirm no empty line
34                 write.append(randomGenerator.nextInt(MAXVALUE)+"");
35                 write.flush();
36                 write.close();
37                 count++;
38             }
39             long writeTime = System.currentTimeMillis()-writeStart;
40             /* write a log file */
41             FileWriter write = new FileWriter(new File("dataLog"));
42             write.write("\n"+count + " file(s) produced;\n"+ NUMSPERFILE
43                 + " integers per file;\n" + MAXVALUE
44                 + " is the max value;\n"+(new Date()).toString()
45                 + "write time = " + writeTime +"\n\n");
46             write.flush();
47             write.close();
48         } catch (IOException e){
49             e.printStackTrace();
50         }
51     }
52 }
53

```

➤ Source Code 2: QuickSort.java

```
1 /*@Purpose: This class simulates quick sort algorithm.
2  @Author:Zhao Xie
3  @Time:9/1/2015
4  @File:QuickSort.java
5 */
6
7 public class QuickSort {
8     /* interface for user*/
9     public void quickSort(int[] collection){
10         quickSort(collection, 0, collection.length-1);
11     }
12     /* quick sort code */
13     private static void quickSort(int[] a, int p, int r)
14     {
15         if(p<r)
16         {
17             int q=partition(a,p,r);
18             quickSort(a,p,q);
19             quickSort(a,q+1,r);
20         }
21     }
22
23     private static int partition(int[] a, int p, int r) {
24
25         int x = a[p];
26         int i = p-1 ;
27         int j = r+1 ;
28
29         while (true) {
30             i++;
31             while ( i< r && a[i] < x)
32                 i++;
33             j--;
34             while (j>p && a[j] > x)
```

```

35         j--;
36
37         if (i < j)
38             swap(a, i, j);
39         else
40             return j;
41     }
42 }
43
44 private static void swap(int[] a, int i, int j) {
45     int temp = a[i];
46     a[i] = a[j];
47     a[j] = temp;
48 }
49 }
50

```

➤ Source Code 3: SortingData.java

```

1 /**
2  * @Purpose: This class will execute internal sort for single
3  *           machine with data file size less than memory size
4  * @Author: Zhao Xie
5  * @Date: 9/1/2015
6  * @File: SortingData.java
7  */
8
9 import java.io.*;
10 import java.util.*;
11 import java.util.Scanner;
12
13 public class SortingData {
14     private static final String FILENAME = "file";
15     private static final int NUMOFFILES = 1;
16     private static final int NUMSPERFILE = 150000000;
17     //private static final int NUMSPERROW = 20;
18     private static final long MAXFILESIZE = 2684354560L;//2.5G
19
20     public static void main(String[] args) throws FileNotFoundException{
21         long programStart = System.currentTimeMillis();
22         int count = 0;
23         long readTime = 0, sortTime = 0,writeTime = 0,programTime;
24
25         try{
26             while(count<NUMOFFILES){
27                 /* reader data from file into array */
28                 long readStart = System.currentTimeMillis();
29                 String newFileName = FILENAME + count;
30                 //check file size
31                 File f = new File(newFileName);
32                 long fileSize = f.length();
33                 if (fileSize<MAXFILESIZE){
34                     int[] A = new int[NUMSPERFILE];

```

```

35 Scanner input = new Scanner(new FileReader(newFileName));
36 for(int i=0;i<NUMSPERFILE;i++){
37     A[i] = input.nextInt();
38 }
39 input.close();
40 readTime += System.currentTimeMillis() - readStart;
41 /* sort the array */
42 long sortStart = System.currentTimeMillis();
43 QuickSort qs = new QuickSort();
44 qs.quickSort(A);
45 sortTime += System.currentTimeMillis() - sortStart;
46 /* write back the file */
47 long writeStart = System.currentTimeMillis();
48 File myFile = new File(newFileName);
49 FileWriter write = new FileWriter(myFile);
50 write.write("");
51 for(int i=0;i<NUMSPERFILE;i++){
52     //if(i%NUMSPERROW==0) {write.append("\n");}
53     write.append(A[i] + "\n");
54 }
55 write.append("\n");

56     write.flush();
57     write.close();
58     writeTime += System.currentTimeMillis() - writeStart;
59 }else{
60     System.out.println("Please use External Sort!");
61     return;
62 }
63 count++;
64 }
65 programTime = System.currentTimeMillis() - programStart;
66 /* write a time log file */
67 FileWriter write = new FileWriter(new File("timeLog"));
68 write.write("\n" + "Program execution time is: " + programTime +
69     " ms;\nRead data time is: " + readTime +
70     " ms;\nSort data time is: " + sortTime +
71     " ms;\nWrite data back time is: " + writeTime +
72     " ms.\n\n");
73     write.flush();
74     write.close();
75 }catch(Exception e){
76     e.printStackTrace();
77 }
78 }
79 }
80

```

➤ Source Code 4: CheckResult.java

```
1 /**
2  * @Purpose This class will check if the sorted file is correct.
3  *       Check the file name before executing.
4  * @Author Zhao Xie
5  * @Date 9/19/2015
6  * @File CheckResult.java
7  */
8 import java.io.BufferedReader;
9 import java.io.FileReader;
10 import java.io.IOException;
11
12 public class CheckResult {
13     private static BufferedReader br;
14     private static String inputFile = "file0";
15
16     public static void main(String[] args) throws IOException{
17         long startCheck = System.currentTimeMillis();
18         br = new BufferedReader(new FileReader(inputFile));
19         String line = br.readLine();
20         int small=0, big;
21         int numCount = 0;
22
23         while(line!=null){
24             if(line.trim().length() == 0) continue;
25             big = Integer.parseInt(line);
26             if(big<small){
27                 System.out.println("The result of sorting is error!");
28                 return;
29             }
30             small = big;
31             line = br.readLine();
32             numCount++;
33         }
34         br.close();
35         System.out.println("Correct!");
36         System.out.println("nums = "+numCount);
37         long time = System.currentTimeMillis() - startCheck;
38         System.out.println("Time used = "+time);
39     }
40 }
41
```


➤ Source Code 5: HadoopSort.java (the version before optimizing)

```

1 /**
2  * @Purpose: Sorting integers in Hadoop cluster for paper lab experiment
3  * @Author:Zhao Xie
4  * @Time:9/30/2015
5  * @File:HadoopSort.java
6  */
7 import java.io.IOException;
8 import java.text.ParseException;
9 import java.util.StringTokenizer;
10
11 import org.apache.hadoop.conf.Configuration;
12 import org.apache.hadoop.fs.Path;
13 import org.apache.hadoop.io.IntWritable;
14 import org.apache.hadoop.io.LongWritable;
15 import org.apache.hadoop.io.NullWritable;
16 import org.apache.hadoop.io.Text;
17 import org.apache.hadoop.mapreduce.Job;
18 import org.apache.hadoop.mapreduce.Mapper;
19 import org.apache.hadoop.mapreduce.Reducer;
20 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
21 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
22
23
24 public class HadoopSort {
25     /**
26      * @Purpose: Decompose input data, and map them into DFS.
27      *          input key/value type is LongWritable/Text, output key/value type
28      *          is IntWritable/Text. Mapper reads value as Text, and the input value
29      *          will be the output key, which is IntWritable type. Actually we don't
30      *          input key and output value.
31      * @probleme: In output file, there is a space printed per line, which wasted write time.
32      */
33     public static class TokenizerMapper
34         extends Mapper<LongWritable, Text, IntWritable, Text>{
35         private final static Text one = new Text();
36         private IntWritable num = new IntWritable();
37
38     public void map(LongWritable key, Text value, Context context
39         ) throws IOException, InterruptedException {
40         String line = value.toString(); //every time read one line from data file
41         if(line.trim().length() != 0){ //avoid empty line lead to error
42             int temp = Integer.parseInt(line); //convert String to int
43             num.set(temp); //only accept int, not Integer type
44             context.write(num, one); //always data in hadoop is key/value pair format
45         }
46     }
47 }
48 /**
49  * @Purpose: Combine the output of mapper into result as expected
50  *          input and output key/value type are same. Facts prove that
51  *          input value can be IntWritable or Text, both of which work.
52  *
53  */
54 public static class IntSumReducer
55     extends Reducer<IntWritable,Text,IntWritable,Text> {
56     private Text result = new Text();
57
58     public void reduce(IntWritable key, Iterable<Text> values,
59         Context context
60         ) throws IOException, InterruptedException {

```

```
61     //int sum = 0;
62     for (Text val : values) {
63         //sum += val.get();
64         context.write(key, result);
65     }
66     // cannot use for(int i;i<sum;i++) loop to control the write operation, not work
67     // cannot use for(Text val : values) second time to do that, not work
68 }
69 }
70
71 public static void main(String[] args) throws Exception {
72     Configuration conf = new Configuration();
73     Job job = Job.getInstance(conf, "hadoop sort");
74     job.setJarByClass(HadoopSort.class);
75     job.setMapperClass(TokenizerMapper.class);
76     job.setCombinerClass(IntSumReducer.class);
77     job.setReducerClass(IntSumReducer.class);
78
79     job.setOutputKeyClass(IntWritable.class);
80     job.setOutputValueClass(Text.class);
81     //the method sets the output value for mapper and reducer
82     //job.setMapOutputKeyClass(IntWritable.class); can only be used to set Map
83     //job.setMapOutputValueClass(IntWritable.class);
84
85     FileInputFormat.addInputPath(job, new Path(args[0]));
86     FileOutputFormat.setOutputPath(job, new Path(args[1]));
87     System.exit(job.waitForCompletion(true) ? 0 : 1);
88 }
89 }
90
```

➤ Source Code 6: HadoopSort.java (the version after optimizing)

```

1 /**
2  * @Purpose: Sorting integers in Hadoop cluster for paper lab experiment
3  * @Author:Zhao Xie
4  * @Time:9/30/2015
5  * @File:HadoopSort.java
6  */
7 import java.io.File;
8 import java.io.FileWriter;
9 import java.io.IOException;
10 import java.text.ParseException;
11 import java.util.StringTokenizer;
12
13 import org.apache.hadoop.conf.Configuration;
14 import org.apache.hadoop.fs.Path;
15 import org.apache.hadoop.io.IntWritable;
16 import org.apache.hadoop.io.LongWritable;
17 import org.apache.hadoop.io.NullWritable;
18 import org.apache.hadoop.io.Text;
19 import org.apache.hadoop.io.compress.*;
20 import org.apache.hadoop.mapreduce.Job;
21 import org.apache.hadoop.mapreduce.Mapper;
22 import org.apache.hadoop.mapreduce.Reducer;
23 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
24 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
25
26 public class HadoopSort {
27     /**
28      * @Purpose: Decompose input data, and map them into DFS.
29      *          input key/value type is LongWritable/Text, output key/value type
30      *          is IntWritable/Text. Mapper reads value as Text, and the input value
31      *          will be the output key, which is IntWritable type. Actually we don't
32      *          input key and output value.
33      * @problem: Fixed one space printed per line;
34      *          still too slow, try to read directly IntWritable for input value on map
35      */
36     public static class TokenizerMapper
37         extends Mapper<LongWritable, Text, IntWritable, NullWritable>{
38         private final static NullWritable one = NullWritable.get();
39         private IntWritable num = new IntWritable();
40
41         public void map(LongWritable key, Text value, Context context
42             ) throws IOException, InterruptedException {
43             String line = value.toString(); //every time read one line from data file
44             //if(line.trim().length() != 0){ //avoid empty line lead to error
45                 int temp = Integer.parseInt(line); //convert String to int
46                 num.set(temp); //only accept int, not Integer type
47                 context.write(num, one); //always data in hadoop is key/value pair format
48             //}
49         }
50     }
51     /**
52      * @Purpose: Combine the output of mapper into result as expected
53      *          input and output key/value type are same. Facts prove that
54      *          input value can be IntWritable or Text, both of which work.
55      *
56      */
57     public static class IntSumReducer
58         extends Reducer<IntWritable,NullWritable,IntWritable,NullWritable> {
59         private final static NullWritable result = NullWritable.get();
60

```

```

61+ public void reduce(IntWritable key, Iterable<NullWritable> values,
62+                 Context context
63+                 ) throws IOException, InterruptedException {
64+     //int sum = 0;
65+     for (NullWritable val : values) {
66+         //sum += val.get();
67+         context.write(key, result);
68+     }
69+     // cannot use for(int i;i<sum;i++) loop to control the write operation, not work
70+     // cannot use for(Text val : values) second time to do that, not work
71+ }
72+ }
73+
74+ public static void main(String[] args) throws Exception {
75+     long startTime = System.currentTimeMillis();
76+     long stopTime;
77+     FileWriter write = new FileWriter(new File("HSTimeLog"));
78+     Configuration conf = new Configuration();
79+     conf.setBoolean(Job.MAP_OUTPUT_COMPRESS,true);
80+     conf.setClass(Job.MAP_OUTPUT_COMPRESS_CODEC,GzipCodec.class,CompressionCodec.class);
81+     Job job = Job.getInstance(conf, "hadoop sort");
82+     boolean status;
83+     try{
84+
85+         job.setJarByClass(HadoopSort.class);
86+         job.setMapperClass(TokenizerMapper.class);
87+         job.setCombinerClass(IntSumReducer.class);
88+         job.setReducerClass(IntSumReducer.class);
89+         job.setMapOutputValueClass(NullWritable.class);
90+
91+         job.setOutputKeyClass(IntWritable.class);
92+         //job.setOutputValueClass(Text.class);
93+         //the method sets the output value for mapper and reducer
94+         //job.setMapOutputKeyClass(IntWritable.class); can only be used to set Map
95+         //job.setMapOutputValueClass(IntWritable.class);
96+
97+         FileInputFormat.addInputPath(job, new Path(args[0]));
98+         FileOutputFormat.setOutputPath(job, new Path(args[1]));
99+
100+     } catch(Exception e){
101+         write.write("\nError Happened. msg: " + e.getMessage() + "\n\n");
102+     }finally{
103+         status = job.waitForCompletion(true);
104+         stopTime = System.currentTimeMillis();
105+         long totalTime = stopTime - startTime;
106+         write.write( "\n" + "start time: " + startTime + "\n" +
107+                    "\n" + "stop time: " + stopTime + "\n" +
108+                    "\n" + "Program execution time is: " + totalTime + " ms;\n\n");
109+         write.close();
110+     }
111+     System.exit(status?0:1);
112+ }
113+ }
114+

```