In Big Data Era: Analysis of Hadoop Cluster Performance

Wedad Alawad
Department of Information Technology, Collage of Computer,
Qassim University,
Buraydah, Saudi Arabia
wmaoad@qu.edu.sa

Awatef Balobaid
Computer Science and Information Technology College
Jazan University
Jazan, Saudi Arabia
asbalobaid@jazanu.edu.sa

2021 International Conference of Women in Data Science at Taif University (WiDSTaif) | 978-1-6654-4948-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/WIDSTAIF52235.2021.9430223

***Abstract*—In recent decades, the dependence on digital tools to assist our daily lives has been growing in almost all sectors and fields. A huge amount of data has been generated and as a result, a new era has emerged: the era of big data. To benefit from this data, several big data analysis tools have emerged, Hadoop being one of them. In this paper, we analyzed Hadoop cluster performance and investigated the effect of the cluster and dataset sizes on its performance. Furthermore, the MapReduce model was used to study and justify its benefits in the big data analysis field. The results showed that the Hadoop cluster and the MapReduce model are powerful in processing and analyzing big data. More benefits can be achieved from the analyzing time perspective when larger data sizes are used. Interestingly, we found that increasing the cluster size beyond a certain point resulted in communication issues, thereby yielding an increase in the CPU time.**

Keywords—big data, Hadoop, MapReduce

I. Introduction

An extensive number of applications have been developed to serve millions of internet users and these applications generate huge amounts of data. Fortunately, this increase in data corresponds with a big reduction in data storage prices, leading to enormous growth in data retention and processing. This large amount of data, known as "big data," is valuable for several fields such as web data mining and natural language processing and highly scalable resources are needed to process and analyze it. One popular approach that enables data analytics by parallel processing is called MapReduce.

MapReduce was presented by Dean et al. in 2004 [1] and has the ability to easily scale out to a cluster of hundreds of commodity nodes and can automatically handle failures in these large settings. Additionally, MapReduce is considered a beneficial resource for people who lack experience in parallel and distributing computing programming because it allows them to process their data in parallel and distributed methods.

The analytics that were performed by using MapReduce in the cloud platform have been profoundly successful on several occasions. For example, *The Washington Post* has used 200 servers on Amazon EC2, which is equal to 1,407 hours of virtual machine time, to convert 17,481 non-searchable PDF pages into WWW format in only nine hours. Furthermore, 100 virtual machines have been used by *The New York Times* to convert 11 million scanned articles to PDFs in just one day [1]. Due to the importance of MapReduce, Amazon web services offers Elastic MapReduce which was built using their compute their storage cloud, EC2 and S3[1][2].

In addition to the cloud platform, MapReduce is also used in the Hadoop cluster. This cluster was designed to provide scalability, large data storage capacity, and increased performance. Overall, it is a remarkably reliable system [3] [4] [5].

In this paper, we used a program that follows the MapReduce programming model to perform an experiment on the Hadoop cluster. We did this for two reasons. The first reason was to learn more about the MapReduce process. The second reason was to study and analyze the effects of the Hadoop cluster size and the input dataset size on the Hadoop cluster performance.

The rest of this paper is organized as follows: Section II discusses the related work. The Hadoop architecture is presented in Section III. Section IV discusses our experiment in detail, including the implementation process, the program that was used, the environment that we have used, and the details of the input datasets. The Hadoop performance from both the CPU time and the wall clock time perspectives is presented and discussed in Section V.

II. Related Work

Several prior studies have explored and analyzed the MapReduce and Hadoop performance from different aspects. Jeffrey Dean and Sanjay Ghemawat performed one of them [1]. In their implementation, a cluster of 1,800 machines and two programs were used to measure MapReduce performance. One of these computations searched through one terabyte of data looking for specific figures. The other program sorted one terabyte of data. They studied the effects of backup tasks on the performance. Furthermore, they killed some tasks to explore the

978-1-6654-4948-9/21/$31.00 ©2021 IEEE

machine failure in MapReduce and analyzed how that affects the program execution time. Additionally, Huang et al. characterized and evaluated the Hadoop framework in terms of speed, throughput, Hadoop Distributed File System (HDFS) bandwidth, system resource utilizations, and data access patterns [6].

III. Hadoop Architecture

Hadoop is an open source implementation of the MapReduce framework. It is typically valuable when the parallel execution for data is required or desired. In fact, data locality optimization is one of the significant features of Hadoop that can reduce the communication overhead. This is usually considered when parallel computing is used by executing the map code on the same DataNode where the data resides. Hadoop has two major components: the MapReduce framework and HDFS. Mapper and Reducer methods are the main parts of the MapReduce programming model; Mapper produces (key/value) pairs from the input data while the reducer combines those pairs to produce the final output. Fig.1 illustrates Hadoop Architecture.

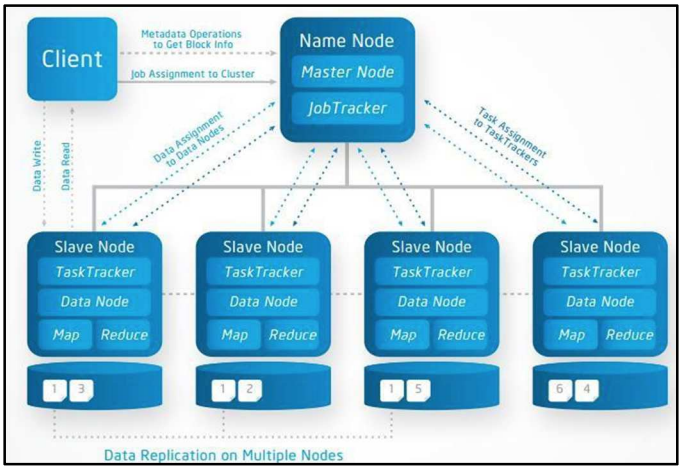

Figure 1: Hadoop Architecture [7].

For clarification, consider a MapReduce process that counts the frequency of word lengths in a book. Each word in the book is taken by a mapper. Then, the mapper generates a (key/value) pair of the form (word length/dummy value). For example, a mapper generates the (5/hello) pair from the input word "hello." After that, a reducer receives the (key/value) pairs that are grouped with the same key, or in other words, the generated pairs for the words that have the same length. Consequently, the reducer will count the number of all the words with lengths "5" when it gets a list that has a key "5" [3][8][9] [10]. An extended example that shows how MapReduce works is discussed in Section III.

While HDFS is a file system used for storing huge amounts of data which runs on clusters of commodity hardware, it is designed to achieve the idea that the most effective data processing pattern is a write-once, read-many-times pattern [9]. In fact, the master/slave architecture is the main architecture of HDFS, and the four components of HDFS are called NameNode, DataNode, JobTracker and Task Tracker [11].

IV. Experiments

To explore Hadoop cluster and MapReduce functionality, we ran a word count program on a Hadoop cluster. In addition to the previous reason, examining and analyzing the effects of the number of nodes and the size of data set on the performance of Hadoop cluster were other reasons that led us to perform this experiment. The word count program that has been used in this implementation is a modified version of an open source word count program which was published in [12]. More information about the original and the modified program will be presented later in this section.

We executed the program 160 times by using two different sizes of data sets and several numbers of nodes. Afterward, the results were presented, analyzed, and discussed. Actually, we edited the code and thus, the new modified code now counts the number of words that start with the same letter of the alphabet. The Hadoop cluster we used is called Zerg Hive. It has a master node, which is the Overlord node, and eight slave nodes. The slave nodes are called Hive1, Hive2, Hive3, and so on. The Hadoop cluster has the following characteristics:

- Operating system: Ubuntu 10.10.
- Apache Hadoop Release 1.0.3
- Sun Java 6.0 Installed
- Overlord: Intel(R) Xeon(TM) CPU 2.40GHz, and the System memory is 1015MiB.
- Hives: Intel(R) Core(TM) 2 Duo CPU, and the system memory is 3274MiB.

In addition to using the Hadoop cluster, to complete our experiment we duplicated some books several times to generate two input datasets; their sizes are 500MB and 1,000MB. [13] is the source of the used books in the experiment. The rest of this section presents and discusses the original word count program and the modified word count program.

A. The Word Count Program

The original word count program is an open source code [12]. Its major idea is counting the frequency of each word in a data set. It consists of three main classes: Mapper, Reducer, and Driver. The Mapper and Reducer phases are considered the two significant stages for the word count program operation. In the mapper phase, the text is tokenized into words and then (key/value) pairs are generated, the key being the word itself, and the value being one [12].

The significant point to be noted here is that the reducer stage starts only after the entire map process is completed [13].

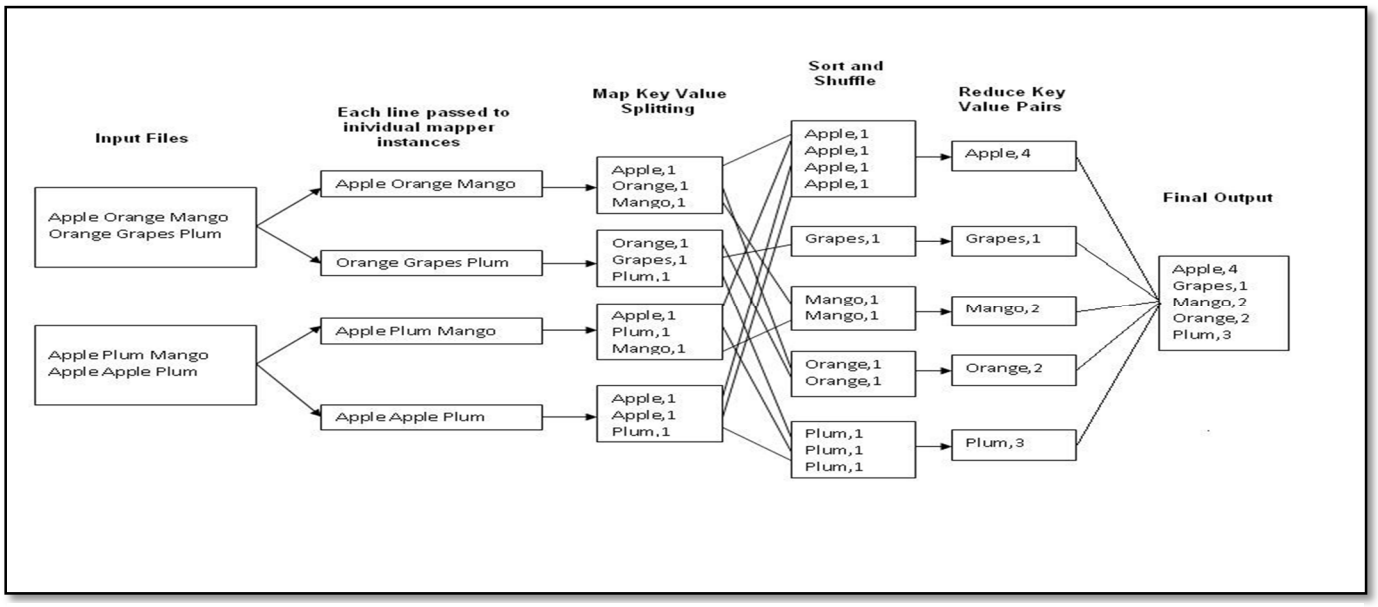

Figure 2:The processes of executing the word count program [13].

Fig. 2 clarifies the executing of the word count program using MapReduce. A sample for the word count program output is shown in Fig. 3.

zrads,	302
zrads.	151
zur	164
zvith	126
zwanzig	196
zweite	121
{	510
{Beginnings	186
{CAMBRIAN	143
{CARBONIFEROUS	170

Figure 3: A sample of the original Word Count Program Output

B. The Modified Word Count Program

Studying the MapReduce model and its role in dealing with big data is consider one of our objectives in this paper. To achieve this, we worked with a large dataset. Furthermore, we modified the original Word Count code for three reasons. First, the original code considers special characters. For example, it may consider a word that starts with special characters different from the same word that does not have a special character prefix. Second, it only counts the frequency of each word in the text, and for large datasets the output will be too long containing each single word in the text. Thus, we believe that counting only the number of words that begin with the same alphabetical letter is reasonable in this case. Third, providing a new code is significant and valuable to the scientific community for reproducibility reasons. The new code counts the number of words that start with the same alphabetical letter. For instance, the output shows how many words start with an "A." Fig. 4 shows a sample for the modified word count program output when the 1,000MB dataset was used.

A	19211101
B	9196567
C	7325549
D	5408730
E	4091205
F	6921324
G	3510454
H	9807840
I	12129415
J	979630

Figure 4: The output for the modified code

V. HADOOP PERFORMANCE

Studying the relationship between the Hadoop performance and the number of nodes that the Hadoop cluster consists of was one of the goals of our experiment. Determining the effects of the input datasets sizes on the performance was another goal as well. Thus, to achieve that, we used the Hadoop cluster that has

eight nodes to execute the modified word count program. In this experiment, the program was applied 80 times by using a 1,000MB dataset and 80 times by using a 500MB dataset.

The experiment began by using all eight nodes to run the code 10 times with the first dataset, and then used the same nodes again to execute the same code 10 times but with the second dataset this time. After that, we reduced the number of nodes by one and repeated the same previous procedures. We kept repeating the same procedures but with a decreased number of nodes each time; the number of nodes decreased each time by one. After each run, the CPU times for Map and Reduce functions were collected and the wall clock times were gathered too.

Running the program several times using the same dataset and number of nodes and calculating the average of all of these runs is important because the communication overhead is not stable in such platforms, (e.g., Cloud and Hadoop platforms). These platforms support fault tolerance and the same data is stored in different nodes. While executing the program, the piece of data needed could be retrieved from any of these nodes, which leads to diversity in the network delay as a result of passing different numbers and types of network components, such as Switches. Additionally, the tasks that are running in the background and sharing the Nodes' CPUs could affect the calculated CPU time.

A. CPU Time Analysis

After applying the program on the 1,000MB dataset, the results showed the average CPU time that the Map and Reduce functions took to complete their execution. They correctly increased by just a few seconds each time the number of nodes increased, as shown in Fig. 5. However, there was a noticeable increase in the time when the number of nodes that were used was greater than four. In fact, the same case is applicable on the 500MB dataset as shown in Fig. 6.

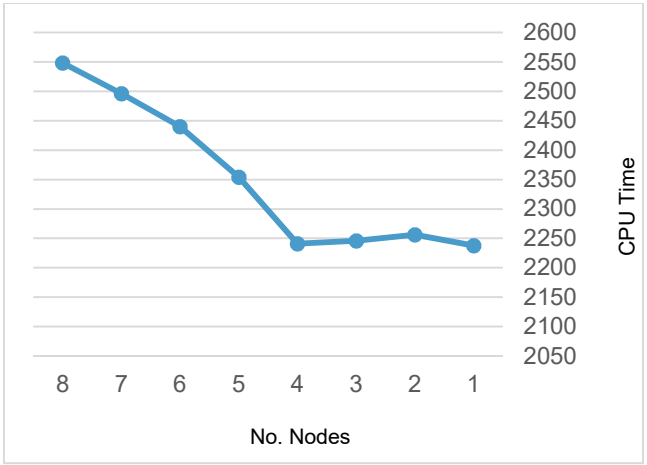

Figure 5: Average of the CPU time for the 1,000MB dataset on 8 different sets of nodes

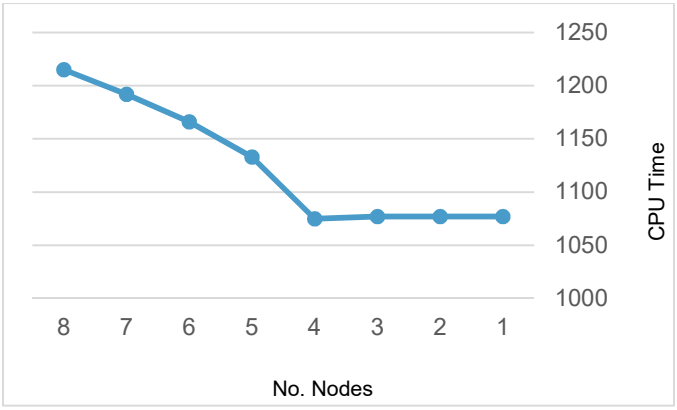

Figure 6: Average of the CPU time for the 500MB dataset on 8 different sets of nodes

B. Wall Clock Analysis

Generally, the differences between the CPU times are very small while the differences between the wall clock times are considerable. See Fig. 7, and Fig. 8. These figures clarify those differences in times when applying the code on 500MB and 1,000MB datasets; in both cases, the wall clock time decreases when the number of nodes increases.

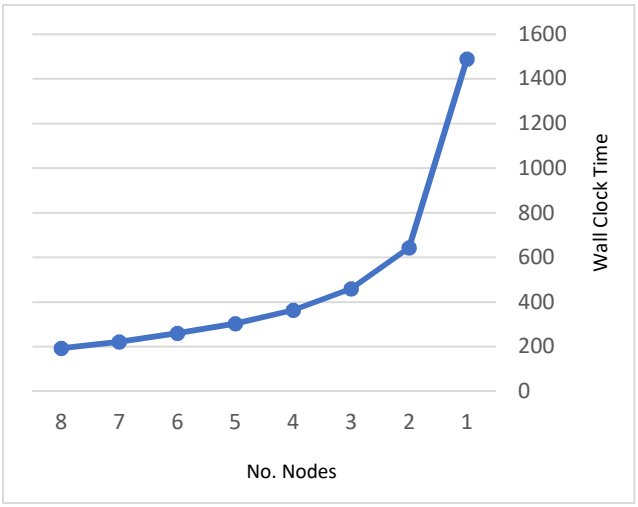

Figure 7: Average of the Wall Clock time for the 500 MB dataset on 8 different sets of nodes

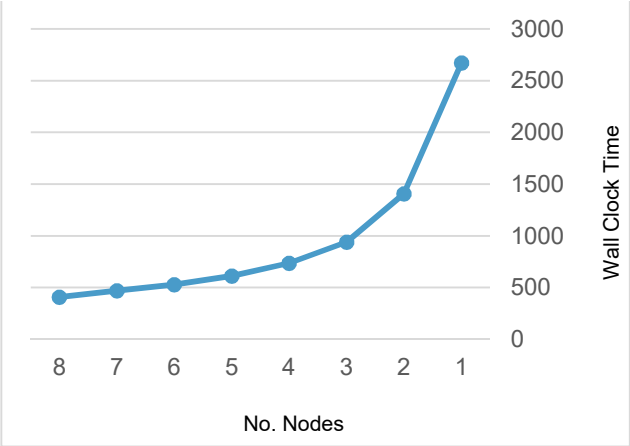

Figure 8: Average of the Wall Clock time for the 1,000MB dataset on 8 different sets of nodes

C. Discussion

To uncover the reason behind the jump in CPU times, we looked at the configuration of the Hadoop cluster that we used to see if there were any communication issues that caused that jump. Our observations directed us to notice an important part in the Hadoop cluster that may have caused that jump. This part is called "Switch" that connects the first four nodes to the second group of four nodes. In our opinion, when we used up to four nodes, only the first four nodes were used in the execution process. Thus, the switch did not play any role in this case; consequently, it did not affect the CPU time and remained almost stable. However, if the number of nodes we used was greater than four, the switch affected the CPU time since it was involved in the execution process. In short, we thought that the communication issues in the Hadoop cluster were responsible for the CPU time increases at a greater ratio than usual.

Fig. 9 shows the CPU time consumed when using the 1,000MB size dataset is double the CPU time when using the 500MB dataset. This is a reasonable result because the number of tasks in the larger dataset is double the number of tasks when the dataset is 500MB. From the wall clock side, it is clear that there is a huge difference in the time between using one node and two nodes in both dataset cases. The ratio of the decrease on the wall clock time when the number of nodes increased was almost constant, except when the number of nodes changed from one to two. In fact, those results clarify the idea of parallel computation and its benefits over the sequential computation. In general, the more nodes we used the more time we saved.

After comparing the wall clock times between different node cases, we compared the wall clock time when executing the code on the 500MB dataset with the wall clock time when using the 1,000MB dataset to see if the times are precisely doubled or not. Fig. 10 shows that the wall clock times are not accurately doubled when applying the code on 1,000MB.

Surprisingly, the wall clock time is less than the doubled time when the size of the input dataset is doubled in most cases; especially when the number of nodes is one. In general, depending on what we have observed, the more data we use, the more time can be saved. Thus, combining several small datasets when possible can be better than using them separately, from the wall clock time perspective.

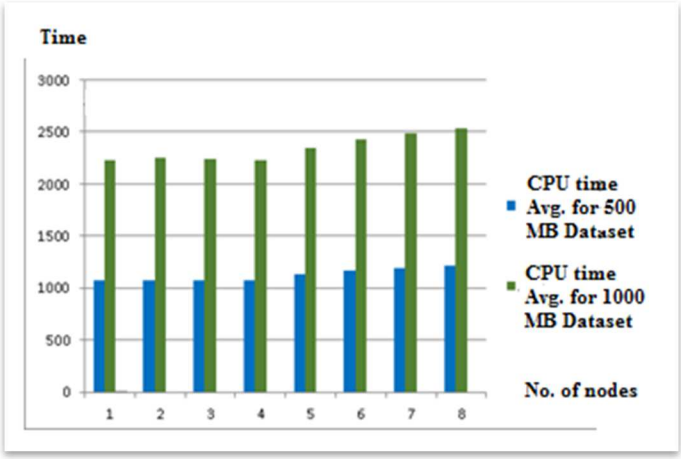

Figure 9: Average CPU time for different datasets on several nodes

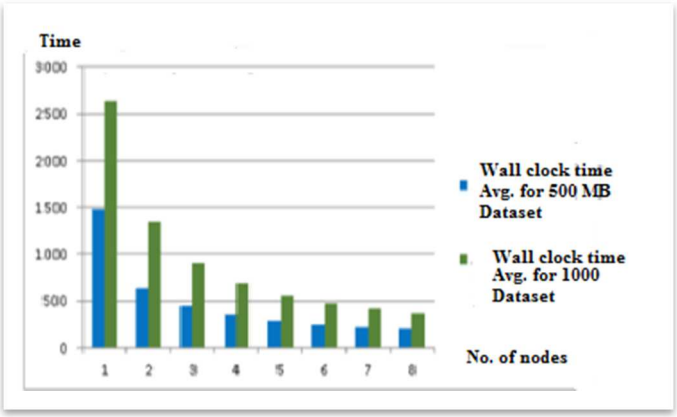

Figure 10: Average Wall Clock time for different datasets on several nodes

VI. Future Work

The findings of this study have raised research questions that will require further examination in future work. An investigation of the effects based on the number of communication devices, such as Switches on the CPU time, is warranted. Additionally, there is a global interest to move toward using visual big data platforms, so we believe that studying the performance of virtual Hadoop clusters and

comparing it to the physical clusters' performance is worth analyzing.

VII. CONCLUSION

Examining the Hadoop cluster performance and the sufficiency of the MapReduce model in analyzing big data was the main purpose of this study. Different factors that may affect the Hadoop cluster performance were investigated, such as the number of nodes in the cluster, the communication between the cluster nodes, and the dataset size. A word count program that followed the MapReduce programming model was executed 160 times on the Hadoop cluster. To conduct these experiments, two datasets and eight different counts of nodes were used.

The results confirmed that using more commodity machines to perform the computation tasks in parallel leads to dramatic decreases in the execution time. However, the communication overhead between the cluster nodes may slightly increase the CPU time. Moreover, the results have shown that additional benefits can be obtained from using the Hadoop cluster when larger datasets are utilized. In sum, the Hadoop cluster platform and the MapReduce programming model are valuable tools for the analyzation of big data.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large Clusters," Communications of the ACM, vol. 51, pp. 137-150. January 2004.

[2] J. Dean and S. Ghemawat,"MapReduce: a flexible data processing tool," Communication of the ACM, vol. 53, pp. 72-77, January 2010.

[3] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in USENIX Workshop on Hot Topics in Cloud Computing (HotCloud09), 2009.

[4] A. Rabkin and R. Katz, "How Hadoop clusters break," IEEE Software, vol. 30, no. 4, pp. 88-94, January 2013.

[5] A. Abuzaid, K. Pawlikkowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads," Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 922-933, August 2009.

[6] S. Huang, J. Huang, J. Dai, T. Xie and B. Huang, "The HiBench Benchmark Suite: characterization of the MapReduce-based data analysis", in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pp. 41-51, 2010.

[7] S. User, *Hadoop Architecture - HDFS and Map Reduce*, January 16, 2016. Accessed on: Mar. 23, 2021. [Online]. Available: http://a4academics.com/tutorials/83-hadoop/835-hadoop-architecture

[8] G. Satya, F. Yuzhang, L. Yang, S. Jin, J. Sun, and K. Rajaraman, "Towards formal modeling and verification of cloud architectures: a case study on Hadoop," IEEE Ninth World Congress on Services, pp. 306-311, June 2013.

[9] T. White, Hadoop: The Definitive Guide, 3rd ed., O'Reilly media, inc. , May. 2012, pp. 17-43.

[10] A. Basirat, and A. Khan, "Introducing an intelligent MapReduce framework for distributed data processing in clouds," IEEE 12th International Symposium on Network Computing and Applications, pp. 61-64, August 2013.

[11] Z. Quan, D. Xiao, C. Tang and C. Rong, "TSHC: Trusted Scheme for Hadoop Cluster," Emerging Intelligent Data and Web Technologies (EIDWT), pp. 344-349. September 2013.

[12] K. Bejoy, *Word Count - Hadoop Map Reduce Example*, April 29, 2011. Accessed on: Dec. 17, 2020. [Online]. Available:http://kickstarthadoop.blogspot.com/2011/04/word-count-hadoop-map-reduce-example.html

[13] M. Noll, *Running Hadoop on Ubuntu Linux (Single-Node Cluster)*, July 5, 2013. Accessed on: Nov. 9, 2020. [Online]. Available: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/#download-example-input-data.