AN ENHANCEMENT LAZY REPLICATION TECHNIQUE FOR HADOOP DISTRIBUTED FILE SYSTEM

Eman S.Abead [1*], Mohamed H. Khafagy[2], Fatma A. Omara[3]

[1]*Faculty of Science, Alasmarya Islamic University, Zliten, Libya*
[2]*Faculty of Computers and Information, Fayoum University, Egypt*
[3]*Faculty of Computers and Information, Cairo University, Egypt*
***Corresponding author:** e.abead@asmarya.edu.ly*

ABSTRACT

The Hadoop Distributed File System (HDFS) is designed to store, analysis, transfer larg datasets reliably, and stream it at high bandwidth to the user applications. HDFS is a variant of the Google File System (GFS). It handles fault tolerance by using data replication, where each data block is replicated and stored on multiple DataNodes. Therefore, the HDFS supports availability and reliability. The existed implementation of the HDFS in Hadoop performs replication in a pipelined manner that takes much time for replication. In this paper, an alternative technique for efficient replica placement, called Enhancement Lazy replication technique, has been proposed. The main principle of this technique is that, the client allows to write a block to two DataNodes in parallel, which store the packet. which will send acknowledgement directly to the client without waiting of receiving acknowledgement from other DataNodes. The experiment has been performed to evaluate the performance of the proposed HDFS replication technique with the default pipelined replication technique and the existed replication techniques; using TestDFSIO benchmark. According to the experimental results (i.e., the execution time and throughput), it is found that the HDFS availability has been improved in the proposed replication technique.

Keywords: Hadoop Distributed File System (HDFS), Pipelined, Replication factor, NameNode, DataNode, Client.

I. INTRODUCTION

Big Data is a concept for massive datasets having more varied, large and complex structure with the difficulties of storing, analyzing and visualizing for processes or results [1]. On the other hands, the need for distributed computing is growing everyday with the increasing of workstations power and the big datasets sizes. Apache Hadoop meets the challenges of Big Data by simplifying the implementation of data intensive and highly parallel distributed applications. Hadoop has been used throughout the world by businesses, universities, and other organizations. It allows analytical tasks to be divided into fragments of work and distributed over thousands of computers, and provides a cost-effective way for storing huge quantities of data. Addition, it provides a reliable and scalable mechanism for processing large amounts of data over a cluster of commodity hardware. And, it provides new and improved analysis techniques that enable sophisticated analytical processing of multi-structured data [2].

The development and implementation of distributed system for Big Data applications is considered a challenge [3, 4]. On other hand, an efficient performance from file system is urgently needed to store the entire internet generated large data and effective handling of huge files. Faster the data transfer means better utilization of distributed system. In recent years, the Hadoop Distributed File System becomes the most popular file system for Big Data Computing due to its availability and fault-tolerance [5]. The HDFS is a file system designed for storing too large files with streaming data access patterns, running on clusters of commodity hardware [6]. The HDFS is considered highly fault-tolerant and is designed to be deployed on low-cost hardware [7]. Also, it provides high throughput access to application data and it is suitable for applications that have big datasets [8]. The HDFS architecture is master/slave. A HDFS cluster consists of a single NameNode, a master server, that manages the file system namespace and regulates the access to files by the clients. Addition, there are a many of DataNodes, usually one per each node in the cluster, which manage the attached storage to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes [8]. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode [7]. When the HDFS client that opens a file for writing, the NameNode will allocate a block with a unique block ID and determines a list of DataNodes to host replicas of that block. The client writes the data block on first the DataNode then the data are pushed to the next DataNode in pipeline form. Stream of bytes are pushed to the pipeline as a sequence of packets. Acknowledgement of data written on DataNodes is also received in pipeline. After all the replicas are written correctly, the client requests NameNode to write the next block(see Fig. 1).

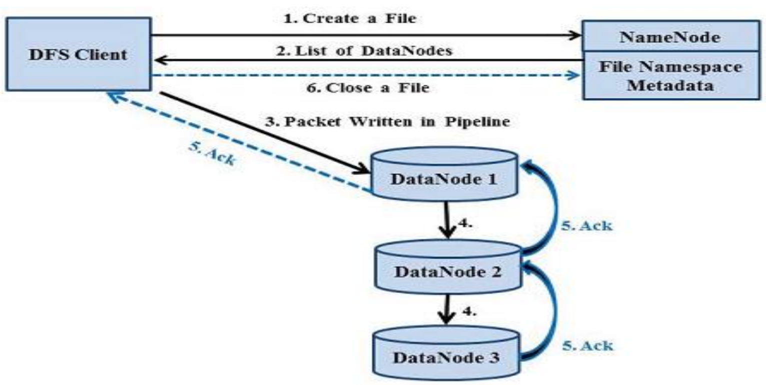

Fig.1. Writing a File on HDFS using pipelined replication technique

These types of pipelined replication scheme decrease the performance of file write operation [5]. Hence, a new proposed replica placement technique has been introduced to improve HDFS availability. According to the proposed technique, HDFS client writes to two DataNodes in parallel, which will send Acknowledgement directly to the client to request another write operation. Then, the two DataNode (DataNode1 and DataNode2) sends replica of the client block simultaneously to two DataNodes in the list, which are used to improve the write performance. Experimental results by using TestDFSIO benchmark prove that, the proposed replication technique can reduce the execution time and increase the write throughput which gives better response time for HDFS client and availability. Based on the performance results, it is found that HDFS write operation throughput falls as file size rises. This is mainly due to the replication factor, limitations of the network bandwidth and file block size. The impact of file blocks size and replication factor on HDFS write performance have also examined from the experimental results.

2. LITERATURE REVIEW

The data replication is a hot research topic in the distributed computing field. There are several mechanisms have been proposed to tackle this issue. For instance, DARE is an adaptive data replication mechanism for HDFS [9]. According to this mechanism, probabilistic sampling and a competitive aging algorithm have been used independently at each node to determine the number of replicas and the location of each replica to be allocated to each file and the location to each replica. DARE mechanism considers the advantage of existing remote data retrievals and selects a subset of the data to be inserted into the file system, hence creating a replica without consuming extra network and computation resources. DiskReduce mechanism is a modification of the HDFS that enables asynchronous encoding of triple replicated data and provides RAID-class redundancy overheads [10]. Addition, to increase a cluster's storage capacity as seen by its users with up to three factors, DiskReduce can delay encoding long enough to deliver the performance benefits of multiple data copies. ERMS [11] has provided a dynamic and elastic data replication mechanism. Based on data access patterns and data popularity, the data in HDFS could be classified into four types; hot data, cooled data, cold data and normal data. Hot data is the popular data. It increases the number of replicas for hot data and cleans up these extra replicas when the data cools down. ERMS shows that it can improve the reliability and performance of HDFS and reduce storage overhead. Qingqing Feng.*et.al*. [12] has introduced Magicube – a high reliable and low redundancy storage architecture for cloud computing with only one replica in HDFS, and an (n, k) algorithm for fault-tolerant. It satisfies both low space overhead and high reliability simultaneously. By executing the fault-tolerant process in the background. Magicube can work well for batch processing jobs. Patel Neha M. *et.al*. [5] has proposed a system which considered an alternative parallel technique for efficient replica placement in HDFS to improve throughput. They proved that HDFS write performance has been improved because client writes all replicas in parallel (see Fig. 2).

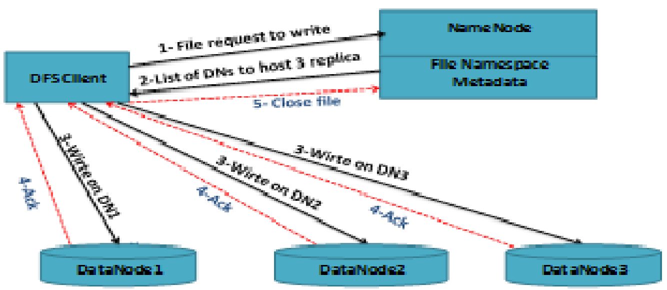

Fig.2. Writing a File on HDFS using Parallel (Broadcast)

Narendra M Patel *.et.al*. [13] has enforced the parallel manner in HDFS, where after requesting NameNode to write a file and receive list of DataNodes to host replica, the client first writes a block to the first DataNode. Once a block is filled in first DataNode, it creates thread and request to DataNode2 and DataNode3 for creating replicas of a desired block in parallel. Once block is written in DataNode2 and DataNode3, they send acknowledgement to first DataNode. After getting acknowledgement from both DataNode2 and DataNode3, DataNode1 sends acknowledgement to the client. Finally, the client sends acknowledgement to NameNode that block is successfully written on three different nodes(see Fig. 3).

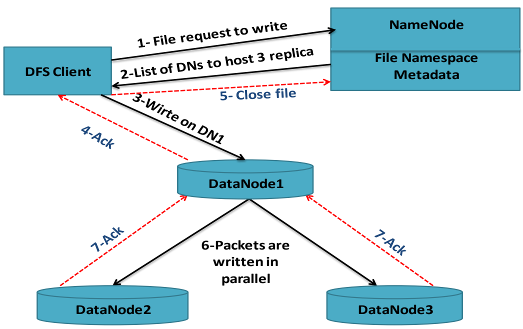

Fig.3. Writing a File on HDFS using Parallel (Master/Slave)

Hong Zhang Patel *.et.al*. [14] has introduced an improved HDFS design called SMARTH. SMARTH utilizes asynchronous multi-pipeline data transfers instead of a single pipeline stop-and-wait mechanism. SMARTH records the actual transfer speed of data blocks and sends this information to the NameNode along with periodic heartbeat messages. The NameNode sorts DataNodes according to their past performance and tracks this information continuously. When a client initiates an upload request, the NameNode will send it a list of DataNodes that it thinks will yield the highest throughput for the client. By choosing higher performance DataNodes relative to each client and by taking advantage of the multi-pipeline design. Eman.S.Abead . *et.al*. [15] has introduced a comparative study of HDFS replication approaches. They provided the comprehensive and theoretical analysis of three existed HDFS replication approaches; the default pipeline approach, parallel (Broadcast) approach and parallel (Master/Slave) approach. The study described the technical specification, features, and specialization for each approach along with its applications. Eman.S.Abead . *et.al*.[20] has enforced The lazy technique has improved HDFS write with respect to the execution time and Throughput. The basic idea behind the lazy technique is to enable the clients to request to write a file in fast time. The high-level overview of the lazy technique and its components has been presented in Fig.4. According to the lazy technique, a single block is written on three different DataNodes; DataNode1, DataNode2 and DataNode3. A Client requests NameNode to write a file. Once a block is filled on DataNode1, DataNode1 sends an acknowledgement to the client. Then, the client sends an acknowledgement to NameNode informing that the block is successfully written on one node; the client can request for next block to write. DataNode1 creates a thread and request DataNode2 and DataNode3 to create replicas of the desired block in parallel. Once, the block is written on DataNode2 and DataNode3; they send an acknowledgement to DataNode1. Finally, After getting acknowledgement from both DataNode2 and DataNode3, DataNode1 sends an acknowledgement to NameNode to write the block that is written in the three DataNodes, DataNode1, DataNode2, and DataNode3, in Metadata.

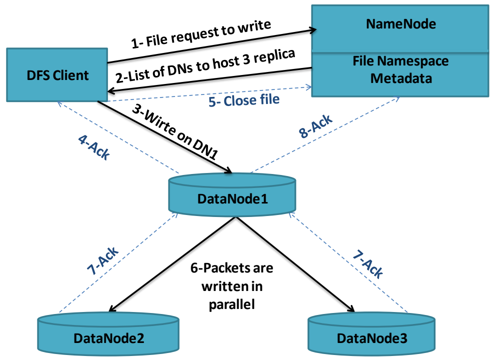

Fig.4. Writing a File on HDFS using Lazy Replication technique

The main drawback of the lazy technique is the single failure problem. This could happen when DataNode1 fails. So, the availability could be affected. The lazy technique has been modified by reconfiguring the DataNodes to overcome this problem called The Reconfigured Lazy Replication Technique (see Fig. 5). That a Client first writes a block to DataNode1 and DataNode2 in parallel, Once a block is filled on DataNode1 and DataNode2, DataNode1 (or DataNode2) sends an acknowledgement to the client. Then, the client sends an acknowledgement to NameNode informing that the block is successfully written on one node; the client can request for next write operation. DataNode1 creates a thread and request DataNode3 to create replicas of the desired block in parallel. Once, the block is written on DataNode3; it sends an acknowledgement to DataNode1. Finally, DataNode1 sends an acknowledgement to NameNode to write the block that is written in the three DataNodes, DataNode1, DataNode2, and DataNode3, in Metadata. If DataNode1 fails to receive an acknowledgement from DataNode3, it resends the same block to it.

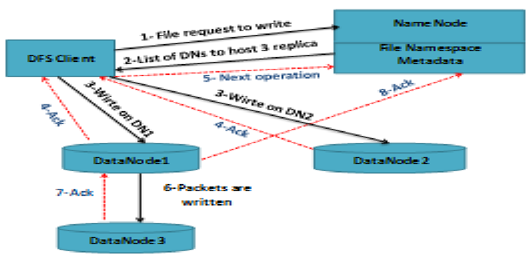

Fig.5. Writing a File on HDFS Using The Reconfigured Lazy Replication Technique

3.THE ANATOMY OF HDFS FILE WRITE PIPELINE AND ENHANCEMENT LAZY REPLICATION TECHNIQUE

Adds data in application to the HDFS by creating a new file then writing the data to it. After the file is closed, the written bytes cannot be modified or removed except that new data can be added to the file by reopening the file for append. HDFS implements a single-writer, multiple-reader model [16]. All HDFS communication protocols are layered on the top of TCP/IP protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. The DataNodes talk to the NameNode by using the DataNode Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol. According to this design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients [8]. The client creates the file by calling *create()* on DistributedFileSystem, Then DistributedFileSystem makes an RPC call to the NameNode to create a new file in the filesystem's namespace, with no blocks associated with it. The NameNode performs many of checks to make sure the file doesn't already exist, and that the client has the permissions to create the file. If these checks pass, the NameNode makes a record of the new file.

DFSOutputStream: creates the files from a stream of bytes. It splits the data into packets; each packet is 64K in size. A packet is broken into chunks. Chunk is 512 bytes and has an associated checksum with it, which it writes to an internal queue, called the data queue.

The DataStreamer streams the packets to the first DataNode in a pipeline, which stores the packet and transmit it to the second DataNode in the pipeline. Similarly, the second DataNode stores the packet and send it to the third (and last) DataNode in the pipeline.

The ResponseProcessor receives an acknowledgement from the DataNodes. When an acknowledgement for a packet is received from all DataNodes, the Response Processor removes the similar packet from the ackQueue [6].

A. HDFS Replica Placement:

The placement of replica is critical issue to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of the rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. Large HDFS instances run on the cluster of computers that commonly spread across many racks. The communication between two nodes in different racks has to go through switches. Mostly, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks. The NameNode identifies the rack id of each DataNode belongs to it via the process outlined in *Hadoop Rack Awareness*. A simple, but the non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows to use bandwidth from multiple racks when reading data. This policy evenly distributes replica in the cluster which makes it easy to load balance on component failure[17,18]. On the other hand, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

Hadoop's default strategy is to place the first replica on the same node as the client (for clients running outside the cluster, a node is random selected, although the system tries not to pick nodes that are too full or too busy). The second replica is placed on a different rack from the first (off-rack), selected randomly. The third replica is placed on the same rack as the second replica, but a different node chosen at random. Further replicas are placed in random nodes in the cluster, although the system tries to avoid placing too many replicas on the same rack [6].

B. Steps to writing a file in HDFS (Pipeline):

HDFS is designed to store reliably very large files across machines in the large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of the file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of a replica of a file. The replication factor can be determined at the file creation time and can be changed later. Files in HDFS are write-once and have one writer strictly at any time.

The steps of writing a file using pipelined replication technique are (see Fig. 3) [16]:

1) HDFS client sends a request to the NameNode to create a new file in the filesystem's namespace.
2) NameNode returns a list of DataNodes to store data block according to replication factor.
3) HDFS client's file data is first divided into blocks with default size and then splits into packets. The list of DataNodes forms a pipeline. Supposing the replication factor is three, so there are three nodes in the pipeline.
4) The packets are sent to the DataNode1 in the pipeline, to be stored and forwarded to the DataNode2 in the pipeline. In the same way, the DataNode2 stores the packet and forwards it to the DataNode3 in the pipeline.

5) Acknowledged by all DataNodes also are received in the pipeline.

6) When the client has finished writing data, it calls *close()* on the stream.This action flushes all the remaining packets to DataNode pipeline and waits for acknowledgments before contacting the NameNode to signal that file is complete.

C. Hadoop DFS file writes operation using Enhancement Lazy Replication Technique:

This section illustrates the components of the Enhancement lazy replication technique. The Enhancement lazy technique has improved HDFS write to enhance the proposed Lazy HDFS replication technique to improve the availability, and in the same time, reduce the execution time and increase write throughput. This has been done by introducing an extra DataNode. The high-level overview of the Enhancement lazy technique and its components has been presented in Fig. 6. The enhancement lazy HDFS technique is implemented as follows:

1) This step as step 1 in both (lazy, configurable) technique, A Client requests NameNode to write a file.

2) This step as step 2 in both (lazy, configurable) technique, The Client first receives a list of DataNodes to write and to host replicas of a single block.

3) This step as step 3 in configurable HDFS technique, The Client first writes a block to DataNode1 and DataNode2 in parallel, which store the packet.

4) Once a block is filled on DataNode1 and DataNode2, DataNode1 and DataNode2 send an acknowledgement to the client. Then, the client sends an acknowledgement to NameNode informing that the

block is successfully written on two nodes; the client can request for next write operation.

5) DataNode1, DataNoe2 creates a thread and request DataNode3, DataNode4 to create replicas of the desired block in parallel.
6) Once, the block is written on DataNode3, DataNode4, it sends an acknowledgement to DataNode1, DataNode2.
7) Finally, After getting acknowledgement from DN3 and DN4, DN1, DN2 sends an acknowledgement to NN to write in Metadata the block that is written in four DataNodes. If DN1, DN2 fails to receive an acknowledgement from DN3 and DN4, it resends the same block to it.

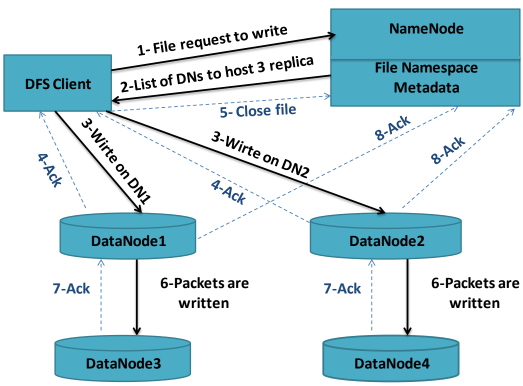

Fig.6. Writing a File on HDFS Using The Enhancement Lazy Replication Technique

4. PERFORMANCE EVALUATION

The performance evaluation of the proposed Enhancement Lazy technique with respect to the pipeline, Lazy and reconfigured lazy techniques is introduced. On the other hands, HDFS write performance is highly dependent on the hardware, network environment, load balancer, and the processing time of each NameNode/DataNodes. Also, the performance may vary as different cluster configuration environment varies.

A. Cluster Configurations.

The proposed Enhancement lazy HDFS replication technique is implemented using a private cluster with one NameNode serves as Metadata storage manager and nine DataNodes provide both computations as MapReduce clients and data storage resources, all commodity computers. All nodes are configured with HCL Intel Core I3 2100, 2.4 GHz processor with 8GB RAM and 320GB SATA HDD. Each node runs Ubuntu 14.10 In all experiments, Hadoop framework 1.2.1 and JDK 1.7.0 is used. These nodes locate in three different racks with Gigabit Ethernet network connecting from Edureka data center.

B. Evaluation Using TestDFSIO.

The TestDFSIO benchmark is used to evaluate the impact of the Enhancement lazy replication technique on HDFS write throughput. This benchmark is a read and writes a test for HDFS. It is helpful for tasks such as stress testing HDFS, to discover the performance bottlenecks in the network, to shake out the hardware, OS and Hadoop setup of the cluster machines (particularly the NameNode and the DataNodes). TestDFSIO measures average throughput for read, write and append operations. TestDFSIO is an application available as part of the Hadoop distribution [19].

Fig 7(a,b) represents the expermintal results for HDFS file write with Replication Factor is four and Block Size is 64MB and with raises of file size from (1,2,3,…,9,10) GB. According to the experiment results in Fig.7(a), it is found that approximately 40% reduction in the execution time of the lazy HDFS replication technique, 25% reduction of the reconfigured lazy replication technique, and 26% reduction of the Enhancement lazy technique comparing to the pipelined replication technique.

Fig.7(b) represents the results of the throughput of the Lazy, reconfigured lazy, Default pipelined and the Enhancement lazy techniques According to the results it has observed that the throughput improvement is around 15% for the lazy HDFS replication technique, 12% for the reconfigured lazy replication technique, and 11% for the Enhancement lazy replication technique comparing to the default pipelined replication. From the results, it is also examined that the throughput is decreased with increasing the file size in the three techniques.

Some factors would affect the HDFS write performance. For example, a file will have fewer blocks if the block size is larger. This can potentially make it possible for the client to read/write more data without connecting with the NameNode, and it also reduces the metadata size of the NameNode, and NameNode workload. This can be necessary for large file systems. On the other hand, larger of the file size, larger of the number of blocks, will increase the total number of the requests from the HDFS clients to NameNode that leads to increase the network overhead.

Actually, HDFS provides flexibility to change default block size using dfs.block.size property. The experiment results are tested with block size=128.

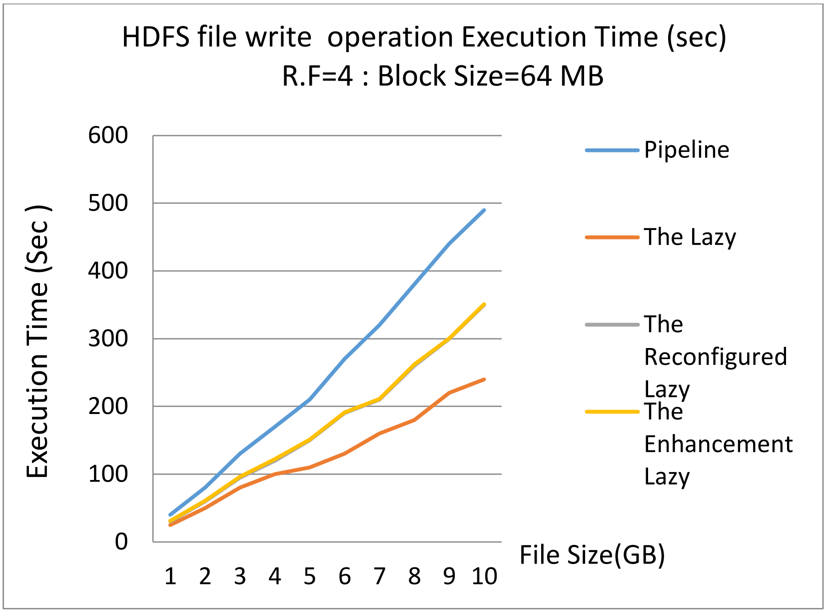

Fig. 7(a) TestDFSIO Execution Time (sec)

note:the Enhancement lazy line is on reconfigured lazy line technique

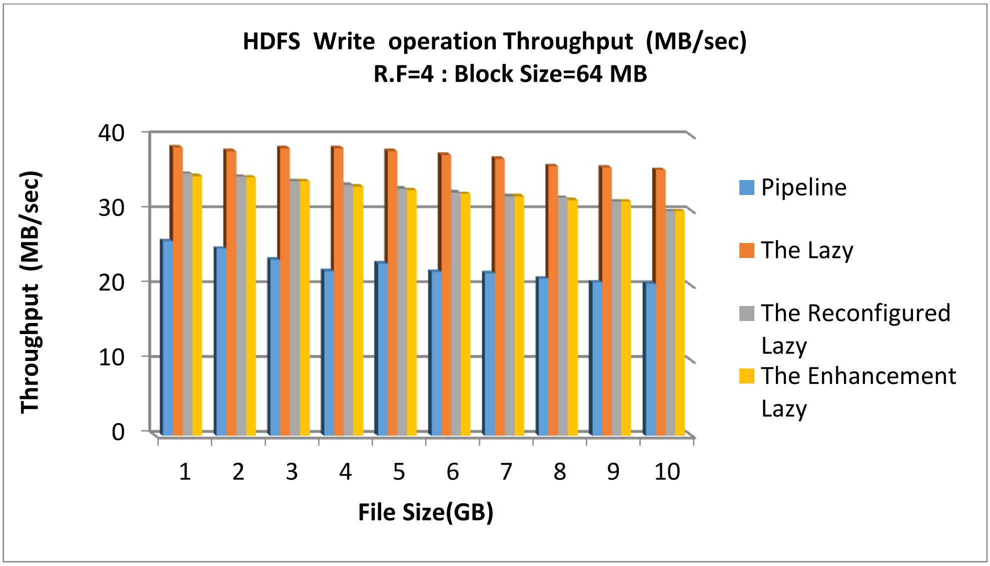

Fig. 7(b) TestDFSIO Throughput (MB/sec)

Fig.7(c) illustrates the performance of the four techniques by considering large block size (i.e., 128 MB). The improvement in file write throughput is approximately 15% to 20% in the lazy technique, approximately 12% to 15% in the reconfigured lazy technique,, and approximately 11% to 16% in Enhancement Lazy replication technique comparing to the pipeline technique. On the other hand, replication factor and the limitation of the network bandwidth also affect the file write throughput.

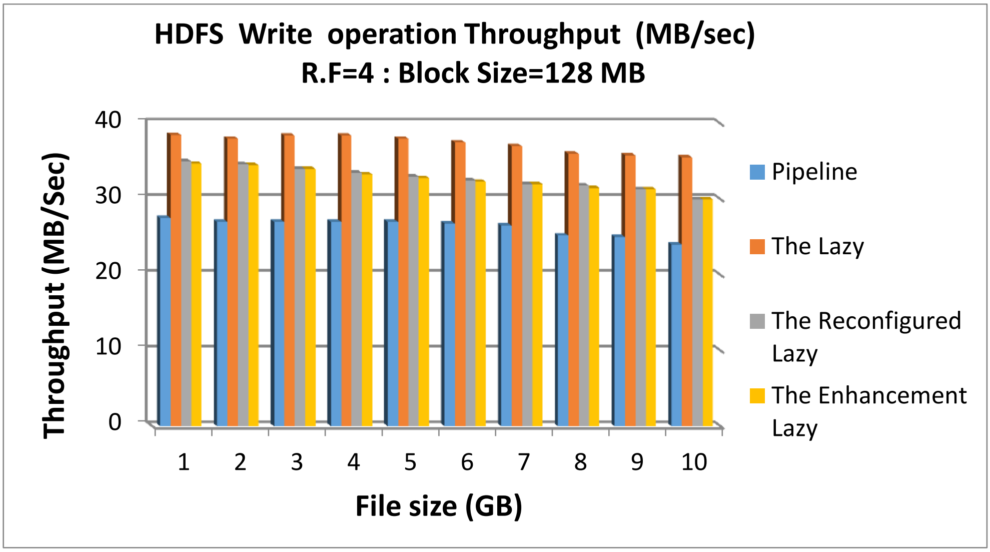

Fig. 7(c) TestDFSIO Throughput (MB/sec) –Different block size

5. CONCLUSIONS

This work builds on and improves the HDFS replication, a technique for decrease wait time for acknowledgements in the HDFS client side and increase data write throughput. Data replication is a technique commonly used to improve data availability and writing throughput in the distributed file systems. In HDFS, each block is replicated on different nodes.

In this paper, the design and implementation of an alternative replication technique called Enhancement lazy replication has been introduced for efficient replica placement on HDFS that can increase availability. By introducing extra DataNode as a backup for the DataNode1 to improve the availability without affecting the write throughput and execution time. According to the implementation results it is found that the reduction of the execution time of the proposed HDFS replication technique relative to pipeline technique shows approximately 25% reduction for HDFS file write with replication factor is three and block size is 64 MB and with raises of file size from (1,2,3,…,9,10) GB using TestDFSIO benchmark. According with the flexibility to change default block size and replication factor the experimental results of throughput with different value of block size and replication factor. With replication factor is 3, Block size is 64MB, It has observed that the throughput improvement is around 12% for the second HDFS replication approach comparing to the default pipelined replication. From the results, it is also examined that the throughput is decreased with increasing the file size in the two techniques. the performance of the two techniques by considering large block size. The improvement in file write throughput is approximately 12% to 15% in second HDFS replication and pipeline techniques. The experiments have been implemented with considering the replication factor of a file two. The experimental results show that the improvement of write throughput for second proposed approach and pipeline approach is up to 18%.

REFERENCES

[1] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, 2013, pp. 42-47.

[2] B. Lublinsky, K. T. Smith, and A. Yakubovich, *Professional Hadoop Solutions*: John Wiley & Sons, 2013.

[3] R. Akerkar, *Big data computing*: CRC Press, 2013.

[4] A. Gkoulalas-Divanis and A. Labbi, *Large-Scale Data Analytics*: Springer, 2014.

[5] M. Patel Neha, M. Patel Narendra, M. I. Hasan, D. Shah Parth, and M. Patel Mayur, "Improving HDFS write performance using efficient replica placement," in *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*, 2014, pp. 36-39.

[6] T. White, *Hadoop: The definitive guide*: " O'Reilly Media, Inc.", 2012.

[7] (Access:25/6/2021 15:00 PM). *HDFS Architecture* Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[8] D. Borthakur, "The Hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[9] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE: Adaptive data replication for efficient cluster scheduling," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, 2011, pp. 159-168.

[10] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: RAID for data-intensive scalable computing," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, 2009, pp. 6-10.

[11] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan, "Erms: an elastic replication management system for hdfs," in *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*, 2012, pp. 32-40.

[12] Q. Feng, J. Han, Y. Gao, and D. Meng, "Magicube: High Reliability and Low Redundancy Storage Architecture for Cloud Computing," in *Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on*, 2012, pp. 89-93.

[13] N. M. Patel, N. M. Patel, M. I. Hasan, and M. M. Patel, "Improving Data Transfer Rate and Throughput of HDFS using Efficient Replica Placement," *International Journal of Computer Applications*, vol. 86, 2014.

[14] H. Zhang, L. Wang, and H. Huang, "SMARTH: Enabling Multi-pipeline Data Transfer in HDFS," in *Parallel Processing (ICPP), 2014 43rd International Conference on*, 2014, pp. 30-39.

[15] Eman.S.Abead, " *An Efficient Replication Technique for Improving availability in Hadoop Distributed File System,**(Unpublished Master's Thesis).**Cairo University, Egypt, 2016, pp 55.*

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-10.

[17] Ebada Sarhan, Atif Ghalwash, Mohamed Khafagy," Queue weighting load-balancing technique for database replication in dynamic content web sites ", Proceedings of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE, 2009, Pp. 50-55

[18] Ahmed M Wahdan Hesham A. Hefny, Mohamed Helmy Khafagy," Comparative Study Load Balance Algorithms for Map Reduce Environment ", International Journal of Applied Information Systems, volume 7, issue 11, 2014, Pp. 41-50

[19] M. G. Noll. (APR 9TH, 2011). *Benchmarking and Stress Testing an Hadoop Cluster With TeraSort*, *TestDFSIO & Co.* (Access: 25/6/2021 15:00 PM) Available: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/