A HADOOP DISTRIBUTED FILE SYSTEM REPLICATION APPROACH

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

[1]*Computer Department, 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

Hadoop Distributed File System (HDFS) is a record framework that is intended to store, examine, and dependably move enormous datasets to client applications. Data replication is utilized to deal with adaptation to handle failures, with every data block being copied and stored on various DataNodes. Thereafter, the HDFS promote availability and reliability. The current Hadoop execution of HDFS does replication in a pipelined design, which consumes most of the daytime. The replication approach is proposed in this concentrate as a substitute methodology for effective replica state of affairs. The basic idea of this procedure is that the client allows two DataNodes to compose one block to the other equally, by storing the package.

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

I. INTRODUCTION

Big Data is a term used to describe huge datasets with a diverse, large and complicated structure that are hard to store, analyses and visualize for processes or results [1]. On the other hand, as workstation capacity and large datasets increase, the need for distributed computation increases daily. Apache Hadoop simplifies the construction of highly parallel, data-intensive systems and the resolution of big data issues. Hadoop is used by institutions, colleges and others worldwide. It can isolate logical cycles in smaller parts and distribute them to many PCs, as well as provide a convenient method to store a lot of information. It also offers a reliable and scalable system for handling massive amounts of data [2].

Creating and deploying distributed systems for big data applications is a challenging task [3, 4]. Other hands, an effective file system is required to store all the massive data created by the Internet and to effectively manage large files. Faster data transport means higher utilization of the distributed system Hadoop Distributed File System has become the most sought-after reporting tool for Big Data Computing in recent years [5]. As a result of its availability and failure tolerance. HDFS is a reporting widget which works on clusters of hardware widgets and is designed to save excessively massive material the use of streaming facts get admission to models [6]. HDFS can be highly fault tolerant and is believed to be connected to low-value hardware [7]. It also offers high-throughput data access, making it excellent for applications with huge data collections [8]. A master/child architecture is used on HDFS. The filesystem namespace is managed by a unique NameNode, commonly called master server, which also controls client access to files within the HDFS cluster. However, many DataNodes, generally one for each node of the cluster, manage storage for the nodes on which it is executed. HDFS shows the filesystem namespace and allows users to record data into files. Internally, it is divided into one or more blocks, which are then stored in a DataNodes collection [8]. Filesystem namespace activities like opening, closing, and renaming files are managed by NameNode.

Open, close, and renaming files and directories are File system namespace activities executed by the NameNode. It also influences how blocks are mapped to DataNodes. The DataNodes are in charge of serving the file system's clients' read and write requests. On the NameNode's instructions, the DataNodes also create, delete, and replicate blocks [7]. When a data

block is first written to a DataNode by a client, the NameNode assigns a block with a special block ID and assigns a list of DataNodes to host copies of that block.. The client first writes the block of data to the DataNode. after which the data is transferred to the next DataNode. As a series many data packet, a stream of bytes is put into the pipeline. The pipeline also receives acknowledgment of the data written to the DataNodes. The client asks the NameNode to write the next block after all replicas have been written appropriately (see Figure 1).

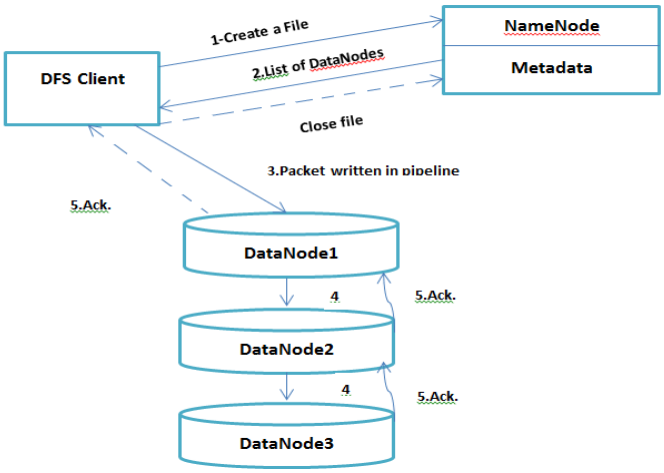

Figure 1: Using a pipelined replication approach to write a file to HDFS

The performance of file writing operations suffers because of these pipeline replication patterns [5]. As a consequence, a new replica placement approach recommended to increase HDFS availability has been introduced. According to the suggested approach, an HDFS client writes simultaneously to two DataNodes, which then send a confirmation to the client, requesting another write operation. The client's block is then sent simultaneously by the two DataNodes (DataNode1 and DataNode2) to the other two DataNodes in the list, thus improving the writing performance. Experimental results using

the TestDFSIO benchmark demonstrate that the suggested replication technique can reduce execution time and improve write throughput, leading to faster and availability times for HDFS client. The effectiveness of written HDFS is demonstrated by the performance figures. The performance figures show that when the file size goes up, the HDFS write rate throughput goes down. Replication, network capability, and file block size restrictions are the main reasons for this. With the help of experimental data, the impact of file block size and replication factor on HDFS write performance was also considered.

II.LITERATURE REVIEW

For distributed computing, data replication is a popular research topic. Several approaches have been proposed for solving this problem. DARE, for instance, is an HDFS replication approach [9]. Based on this process, each node independently employed probability sampling and a competitive aging algorithm to select the number of replicates and the position of each replicate to be assigned to each file and replicate. The DARE method takes into account the advantages of recovering existing remote data and selected parts of the data to be inserted into the filesystem, leading to the creation of a replica without the use of additional network and computational resources. The DiskReduce approach is an HDFS improvement that allows for asynchronous encoding of triple replicated data and provides RAID-class redundancy overheads [10]. Moreover, DiskReduce can delay encoding long enough to give the performance advantages of repeated data copies, increasing a cluster's storage capacity by up to three times, according to users. The reproduction of ERMS data [11] is dynamic and elastic. Data in HDFS can be divided into four groups based on data access models and

their popularity: hot data, cooled data, cold data, and normal data. Popular data is referred to as hot data. When the data is hot, it increases the number of replicas and cleans up additional replicas when the data cools down. ERMS has demonstrated its potential to improve the quality of life. Magicube is a high-reliability, low-redundancy cloud storage architecture with a single HDFS replica and a fault tolerant method (n, k), according to Qingqing Feng.et.al. [12]. It simultaneously meets space-saving requirements and high reliability. The process of fault tolerance is executed in the background. Magicube is a right choice for batch processing. To increase throughput, Patel Neha M. et al. [5] suggested a system which explored a parallel approach to effectively HDFS replication. They demonstrated that HDFS writing speed improved due to the client's writing all replicates simultaneously (see Fig. 2).

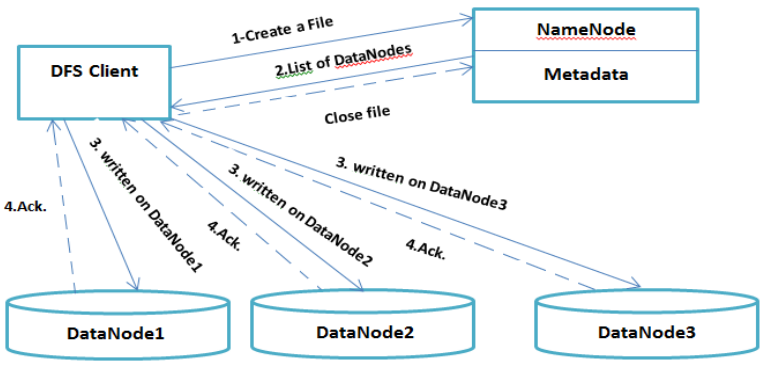

Figure.2. Using Parallel approach to Write a File on HDFS

The client sends a block to the first DataNode after having instructed NameNode to create a file and obtaining a DataNode list to allow the reply, according to Narendra M Patel et al. [13].When a block is populated in the first DataNode, it starts a thread and sends a request to DataNode2 and

DataNode3 to generate simultaneous replicates of the required block. Once DataNode2 and DataNode3 have written a block, they acknowledge receipt on the first date. Upon receipt of the acknowledgement of DataNode2 and DataNode3, DataNode1 sends an acknowledgement of receipt to the client. Finally, NameNode receives a notification from the client that the block was in fact written on three nodes. (see Fig. 3).

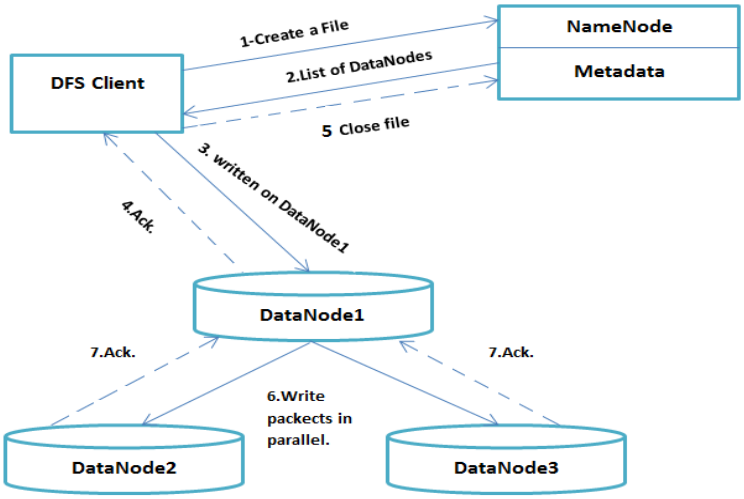

Figure.3. Using a parallel (Master/Slave) approach to write a file on HDFS

SMARTH is an enhanced HDFS concept brought forward by Hong Zhang Patel et al. [14]. Rather than using a single pipeline standby and shutdown approach, SMARTH uses asynchronous multi pipeline data transfers. SMARTH maintains a record of the actual transfer speed of the data block and sends periodic heartbeat messages to the NameNode. DataNodes are sorted by NameNode based on their historical performance, which it keeps monitoring in real time. When a client requests a download, the NameNode sends the client a DataNodes list that it believes will provide the best throughput. Using multi pipeline design and selecting best performing

DataNodes relative to each client. Eman.S.Abead. et.al.[20] has made it mandatory. In terms of execution time and throughput, the lazy method has enhanced HDFS write. The lazy approach's main goal is to let clients request that a file be written in a short period of time. The sluggish technique and its components are depicted in Figure 4 as a high-level overview. A single block is written to three separate DataNodes using the lazy approach: DataNode1, DataNode2, and DataNode3. NameNode is asked to write a file by a client. When a block on DataNode1 is full, DataNode1 gives the client an acknowledgement. The client then gives NameNode an acknowledgement that the block has been successfully sent to a node; the client can then request to write the next block. DataNode 1 begins a thread and requests simultaneous replicas of the desired block from DataNodes 2 and 3. When the block is overwritten on DataNode2 and DataNode3, they send a message to DataNode1. After receiving the acknowledgment for DataNode2 and DataNode3, DataNode1 sends an acknowledgment to NameNode. This allows NameNode to write the block in Metadata which is written by the three DataNodes, DataNode1, DataNode2 and DataNode3.

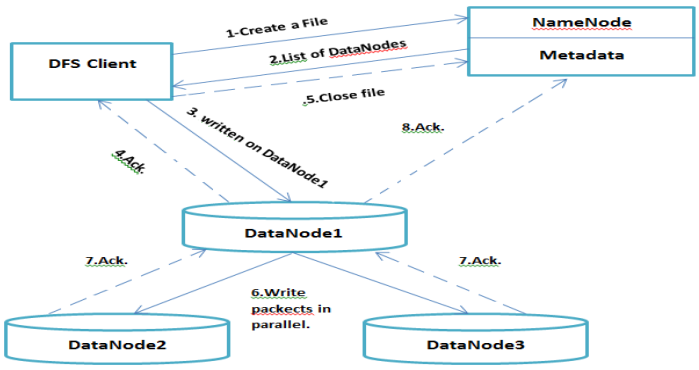

Figure.4 Using the Lazy Replication approach to write a file to HDFS

The single failure problem is the fundamental disadvantage of the lazy approach. This may occur if DataNode1 fails. As a result, availability may be impacted. To tackle this difficulty, the lazy approach has been changed by reconfiguring the DataNodes, which is known as The Reconfigured Lazy Replication approach (see Fig. 5). When a client uploads a block to both DataNode1 and DataNode2, DataNode1 (or DataNode2) acknowledges the client after the block is filled on both DataNode1 and DataNode2. The client then gives an acknowledgement to NameNode indicating that the block was successfully written to a node and that the client is now ready to request the next write operation. DataNode1 starts a thread and requests DataNode3 to create parallel replicates of the desired block. Once the block is written to DataNode3, it acknowledges receiving the DataNode1 with a message. Finally, DataNode1 accepts NameNode's request to write the block into Metadata which is written into DataNode1, DataNode2 and DataNode3. If DataNode1 does not receive a receipt, the same block is sent back to DataNode3.

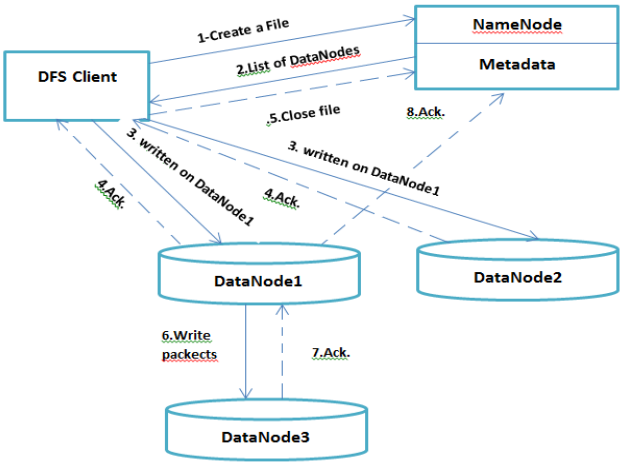

Figure.5. Using the Reconfigured Lazy Replication approach to write a file to HDFS

III. AN ANATOMICAL AND REPLICATION APPROACH FOR THE HDFS FILE WRITING PIPELINE

Creates a new file and writes the data to it to add data from the application to the HDFS. The written bytes cannot be updated or erased after the file is

closed; However, by reopening the file for append, new data can be inserted. HDFS employs a single-writer, multiple-reader design. [16]. The TCP/IP protocol is placed on top of all HDFS communication mechanisms. A client connects to the NameNode computer over a customizable TCP port. The DataNode Protocol is used to communicate with the NameNode. The Client Protocol and the DataNode Protocol are both wrapped in a Remote Procedure Call (RPC) abstraction. The NameNode never initiates any RPCs as a result of this architecture. It instead only reacts to RPC requests from DataNodes and clients [8]. The client creates the file with a request to create() on DistributedFileSystem.which then sends an RPC query to NameNode to create a new file without a block in the filesystem namespace. Many controls are performed by the NameNode to make sure that the file does not already exist and that the client has the necessary permissions to create it. NameNode creates a record of the new file if it passes.

DFSOutputStream: DFSOutputStream: Creates files from the byte stream. Data separates into packets of 64 kilobytes apiece. The contents of a packet are cut into bits. The checksum is written to an internal queue known as the data queue, and chunks are 512 bytes long.

The DataStreamer The DataStreamer sends packets to the first DataNode in the pipeline, which stores them and sends them to the second DataNode in the pipeline. Similarly, before transmitting the packet to the pipeline's third (and final) DataNode, the second DataNode caches it.

The ResponseProcessor The ResponseProcessor receives an acknowledgement from the DataNodes. When the response processor receives an acknowledgment for a package from all DataNodes, it deletes the similar package from the ackQueue [6].

A. Positioning of HDFS Replication

The location of replicas is crucial to HDFS speed and reliability. HDFS differs from most other distributed file systems in that it optimizes replica placement. The Rack Replica Placement Policy is designed to enhance data reliability, availability and usage of network bandwidth. Important HDFS instances are usually executed on a cluster of machines spread over several racks. Communication between nodes in various racks necessitates the use of switches. Network bandwidth between machines in a single rack is often higher than between machines in other racks. The Hadoop Rack Awareness technique is used by the NameNode to get the rack ID for each DataNode that belonging to it. Placing clones on separate racks is a straightforward but inefficient strategy. When an entire rack fails, this eliminates data loss and allows data to be read using bandwidth from neighboring racks. This approach evenly distributes replicas across the cluster, allowing for simple load balancing in the event of component failure[17]. This strategy, However, increases the writting cost by demanding a write to transfer blocks to numerous racks.

By default, Hadoop sets the initial copy to the same node as the client (for clients running outside the cluster, one node is randomly selected, although the system tries not to select nodes which are too full or too busy). The second replica is placed on a rack different from the first randomly (off-rack). The third duplicate goes on the same rack as the second, but at a different node. Other replicas are deployed at random nodes around the cluster, the algorithm trying to prevent placing too many replicas on a single rack [6].

B. Using HDFS (Pipeline) to write a file

HDFS is designed for reliable storage of extremely large files on multiple devices in a huge cluster. Except for the last block, each file is recorded as a series of blocks, all of which have the same size. The blocks in the file are replicated for failure tolerance. Block size and replication factor may be changed per file. An application is able to specify the number of clones in a file. The replication factor can be defined during file creation and later modified. HDFS Write-once files contain only one recorder at all times.

The following are the stages involved in writing a file utilizing the pipelined replication approach (see Fig. 3) [16]:

1) The HDFS client requests the NameNode to create a new file in the file system naming space.
2) NameNode gives a list of DataNodes where data blocks can be stored based on the replication factor.
3) The packets are transmitted to DataNode1 in the pipeline, where they are stored and transferred to DataNode2. The DataNode2 saves the packet and transmits it to the DataNode3 in the pipeline in the same way.
4) The packets are transmitted to DataNode1 in the pipeline, where they are stored and transferred to DataNode2. The DataNode2 saves the packet and transmits it to the DataNode3 in the pipeline in the same way.
5) DataNodes for which all DataNodes have acknowledged receipt are also received within the pipeline.

6) When finish client has writing data in the flow, it uses close().

C. Replication-assisted Hadoop DFS file write operation

The following section describes the components of the replication approach. The proposed HDFS replication approach enhanced HDFS write performance while reducing execution time and enhancing availability of the proposed Lazy HDFS replication approach. An extra DataNode was added to do this. Figure 6 provides an overall view of the replication strategy and its components. The following is the HDFS replication strategy:

1) In this phase, a client instructs NameNode to write a file, which corresponds to step 1 of both approaches (lazy, configurable). The client first receives a list of DataNodes to write and host single-block replicas, similar to step 2 in both approaches (sloth, customizable).

2) The client writes a block to DataNode1 and DataNode2, that store the package in a parallel approach, similar to step 3 of the configurable HDFS approach.

3) Once DataNode1 and DataNode2 have completed a block, DataNode1 and DataNode2 send a confirmation to the client. After that, client acknowledges receipt to NameNode, indicating that the block has been written on two nodes and that the client is now ready to request the next write operation.

4) DataNode1 and DataNode2 start a thread and request that DataNode3, DataNode4, and DataNode5 all make clones of the desired block at the same time.

5) Once the block is written into DataNode3, DataNode4, the block sends an acknowledgement of receipt to DataNode1, DataNode2.

6) Finally, DN1 sends an acknowledgement to NN, requesting that the block written in four DataNodes be written in Metadata, after gaining approval from DN3 and DN4. If DN1 does not receive a response from DN3 or DN4, DN2 sends the same block to them once more.

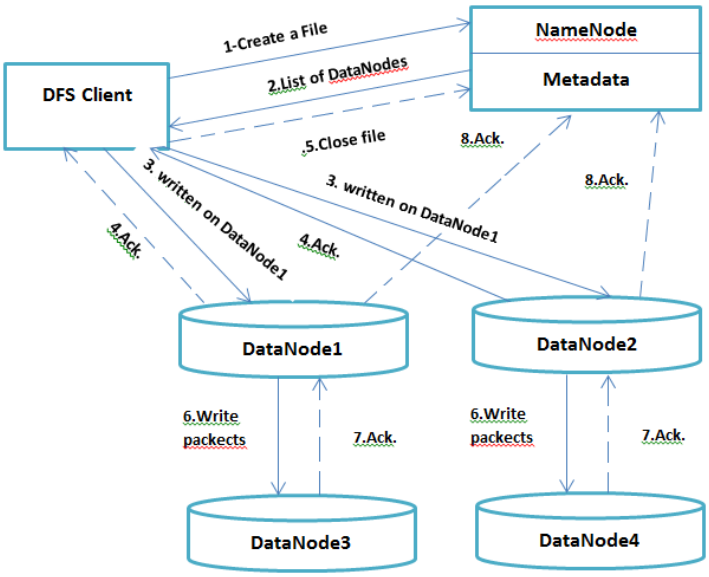

Figure.6. Using the Replication Approach to Write a File on HDFS

IV. PERFORMANCE EVALUATION

The suggested performance evaluation for replication The pipeline concept is introduced, as well as the lazy and modified lazy approaches. The hardware, network environment, load balancer, and processing time of each NameNode/DataNode, on the other hand, have a significant impact on

HDFS write performance. Furthermore, depending on the cluster configuration option, performance may vary.

A. Configurations of Cluster

The recommended HDFS replication approach is carried out within a private cluster with a running NameNode, the metadata storage manager and nine DataNodes, all base machines, offering both compute and data storage resources as MapReduce client. All nodes have an HCL Intel Core I3 2100 2.4 GHz processor, 8GB RAM, and a 320GB SATA HDD. Every device has Ubuntu 14.10 installed. In all studies, Hadoop frame 1.2.1 and JDK 1.7.0 are used. These nodes are organized in three racks and connected to the Edureka data center over a Gigabit Ethernet network.

B. Evaluation Using TestDFSIO

The effect of the suggested HDFS replication approach on the HDFS write throughput is evaluated using the TestDFSIO benchmarking. This HDFS benchmark is for reading and writing test. It's useful for such things as HDFS stress testing, discovering network performance bottlenecks, and shaking up cluster machine hardware, OS, and Hadoop configuration (especially the NameNode and the DataNodes). The mean read, write, and append throughput for is measured by TestDFSIO. TestDFSIO is a software supplied with the Hadoop release [19].

Figure 7(a,b) shows experimental results for writing HDFS files with a four-fold replication factor and a 64 MB block size, as well as file sizes ranging from one to ten gigabytes. According to the trial results in Fig.7(a), the lazy HDFS replication approach has a 40% reduction in execution time, a 25% reduction in the reconfigured lazy replication approach, and a 26%

reduction in the suggested HDFS replication approach when compared to the pipelined replication approach.

The results of the throughput of the Lazy, modified lazy, Default pipelined, and the suggested HDFS replication approach are shown in Fig.7(b). According to the findings, the lazy HDFS replication approach has a throughput improvement of around 15%, the reconfigured lazy replication approach has a throughput improvement of 12%, and the The suggested HDFS replication approach has a throughput improvement of 11% versus the default pipeline replicate. The results also show that when the file size grows larger, the throughput decreases in all three strategies.

A variety of factors influence the write performance of HDFS. A file will have less blocks if the block size is greater, for instance. This allows the client to read/write more data without connecting to NameNode, and also reduces the size and weight of NameNode metadata. This may be important for large file systems. Larger file sizes and block counts, on the other hand, will increase the overall number of HDFS client requests to NameNode, resulting in additional network overhead.

Actually, the dfs.block.size property in HDFS allows you to adjust the default block size. The results of the experiment are validated with a block size of 128.

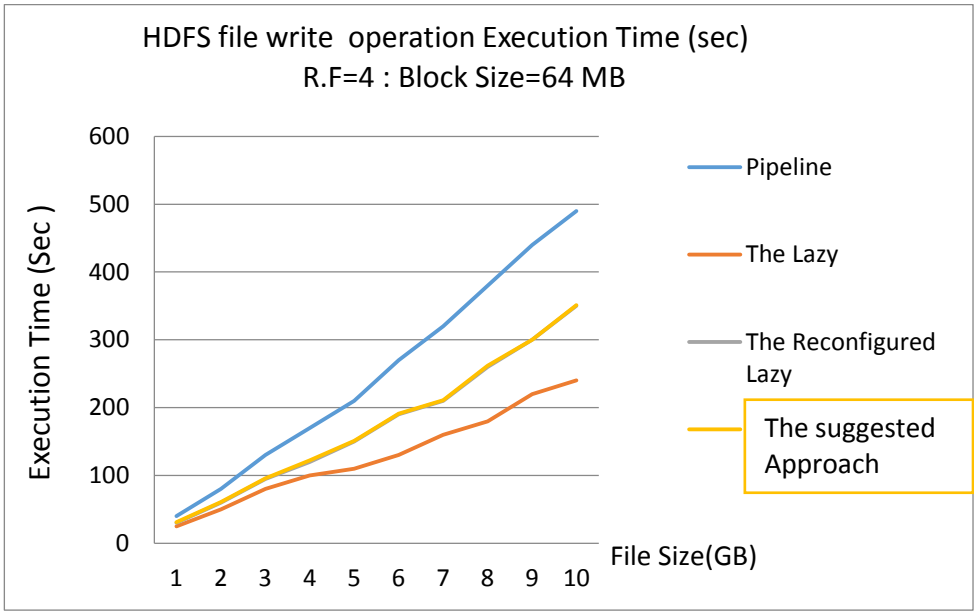

Fig. 7(a) TestDFSIO Execution Time (sec)

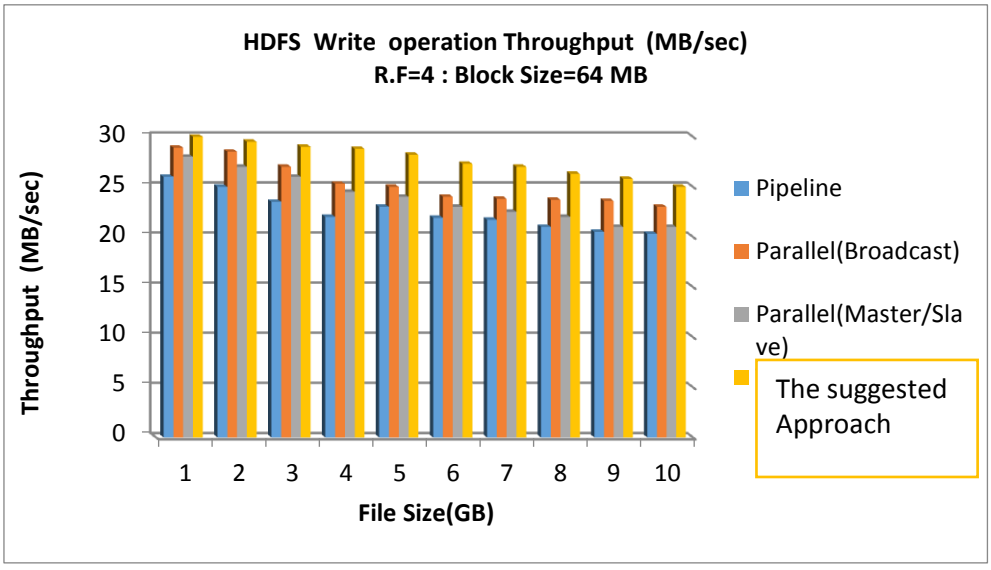

Fig. 7(b) TestDFSIO Throughput (MB/sec)

When considering a large of block size, Figure 7(c) shows the performance of the four approaches (i.e., 128 MB).Compared to the pipeline technique, the lazy strategy improves throughput of file write by about 15% to 20%, the modified lazy approach of about 12–15%, and the proposed HDFS replication approach by about 11–16%. On the other side, file write throughput is affected by replication factor and network bandwidth limits.

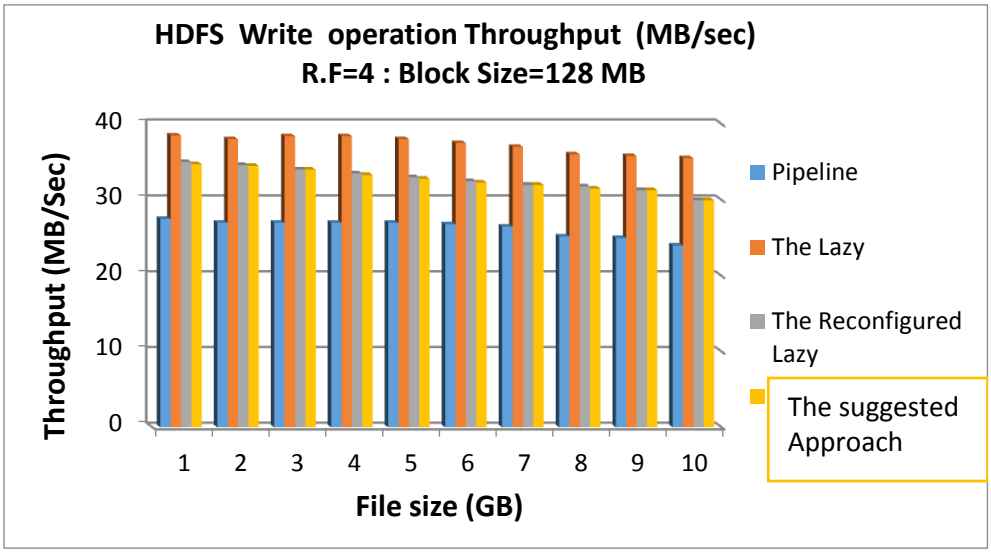

Fig. 7(c) TestDFSIO Throughput (MB/sec) –Different block size

V.CONCLUSIONS

This work expands on and enhances HDFS replication, a strategy for reducing acknowledgement wait times and increasing data write throughput on the HDFS client side. Data replication is a approach that is often used in distributed file systems to improve data availability and writing throughput. Each block in HDFS is replicated on many nodes. The design and implementation of an alternative replication approach known as The suggested HDFS replication approach for efficient replica placement on

HDFS that can increase availability are presented in this research. by adding an additional DataNode as a backup for DataNode1 to increase availability without compromising write performance or execution time. According to the implementation results, the proposed HDFS replication approach reduces the execution time of HDFS file writes by about 25% compared to pipeline approaches when the replication factor is three and the block size is 64 MB, and when the file size is increased from (1,2,3,...,9,10) GB using the TestDFSIO benchmark. The experimental findings of throughput with different values of block size and replication factor are based on the flexibility to modify default block size and replication factor. When the replication factor is 3, and the block size is 64MB, the throughput improvement for the second HDFS replication strategy is roughly 12% when compared to the default pipelined replication. As a result of the findings, The results also show that when the file size grows larger, the throughput of the two approaches decreases. By taking huge block size into account, the performance of the two strategies may be compared. In second HDFS replication and pipeline approaches, the increase in file write throughput is approximately 12 to 15%. The experiments were carried out with the replication factor of a file two in mind. The testing results suggest that the second recommended strategy and pipeline approach boost write throughput by up to 18%.

REFERENCES

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

[2] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, 2013, pp. 43-48.

[3] R. Akerkar, *Big data computing*: CRC Press, 2013, pp. 25-55.

[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. 35-38.

[6] (Access:25/5/2022 15:00 PM). *HDFS Architecture* Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

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

[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. 158-169.

[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. 5-10.

[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. 33-40.

[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. 88-94.

[13] 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.

[14] 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.

[15] Eman.S.Abead, Mohamed H. Khafagy, and Fatma A. Omara, "A Comparative Study of HDFS Replication Approaches,", the International Journal of IT and Engineering Issues, Vol. 03, Issue-08, August 2015, pp 5-11

[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

[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/

[20] Eman.S.Abead, Mohamed H. Khafagy, and Fatma A. Omara, " An Efficient Replication Technique for Hadoop Distributed File System, in Proceeding of the International Journal of Scientific and Engineering Research, Volume 7, Issue 1, ISSN: 2229-5518, January 2016, pp 254-261 .