An Efficient Replication Technique for Hadoop Distributed File System

Eman S.Abead
Faculty of Computers and Information
Cairo University, Egypt
emanabead@gmail.com

Mohamed H. Khafagy
Faculty of Computers and Information
Fayoum University, Egypt
mhk00@fayoum.edu.eg

Fatma A. Omara
Faculty of Computers and Information
Cairo University, Egypt
f.omara@fci-cu.edu.eg

Abstract—**The Hadoop Distributed File System (HDFS) is designed to store, analysis, transfer massive data sets 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 reliability and availability. 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 Lazy replication technique, has been proposed. The main principle of this technique is that, the client allows to write a block to the first DataNode, which will send acknowledgement directly to the client without waiting of receiving acknowledgement form other DataNodes. The proposed technique has been implemented into two versions; Lazy and Reconfigurable Lazy. 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; parallel (Broadcast) and parallel (Master/Slave) using TestDFSIO benchmark. According to the experimental results, it is found that the HDFS write throughput has been improved up to 15% in the proposed replication technique.**

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

I. INTRODUCTION

Big Data is a term for massive data sets having large, more various and complex structure with the difficulties of storing, analyzing and visualizing for more of processes or results [1]. On the other hands, the need for distributed computing is growing every day with the increasing of workstations power and the data sets sizes. On the other hand, 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. On the other hands, it provides a cost-effective way for storing huge set of data , and 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. Also, it provides a scalable and reliable mechanism for processing large amounts of data over over a cluster of commodity hardware to process large amount of data. Also, it provides new analysis techniques that enable sophisticated analytical processing of multi-structured data [2].

The development and implementation of distributed system for Big Data applications are considered a challenge [3, 4]. In data era, an efficient performance from the file system is urgently needed to store the entire large data that would be generated through the internet and efficiently handle huge files. The faster the data transfer means better utilization of distributed system. In the recent years, the HDFS becomes the most popular file system for Big Data Computing due to its availability and fault-tolerance [5]. The HDFS is a file system that is designed for storing very large files with streaming data access patterns, running on the 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 large data sets [8]. The HDFS architecture is master/slave. An 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. Also, there are some DataNodes, usually one per each node in the cluster, which manage the attached storage to the nodes that they run on. The 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 open, close, and rename files and directories. Also, It determines the mapping of blocks to the DataNodes. The DataNodes are responsible for serving the read and the write requests from the file system's clients. Also, the DataNodes perform the block creation, deletion, and replication across instruction from the NameNode [7]. According to the HDFS in Hadoop, the HDFS client opens a file for writing where 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 the first DataNode, and then the data are pushed to the next DataNode in pipeline form. A stream of bytes is pushed to the pipeline as a sequence of packets. Acknowledgement of data written on DataNodes is also received in the pipeline. After all the replicas are written correctly, the client requests NameNode to write the next block (see Fig. 1).

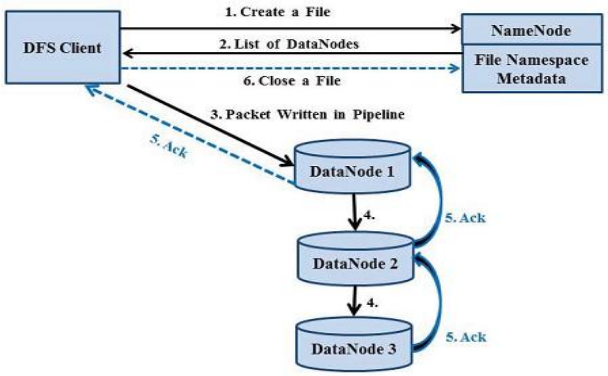

Fig.1. Writing a File on HDFS using pipelined replication technique

This kind of pipelined replication scheme affects the performance of file write operation because of the time overhead [5]. Hence, a new lazy replica placement technique has been introduced to improve HDFS write throughput. According to the lazy technique, HDFS client writes to first DataNode, which will send Acknowledgement directly to the client to request another write operation. Then, the first DataNode sends a replica of the client block simultaneously to two DataNodes on the list. The experimental results using TestDFSIO benchmark prove that, the lazy replication technique can reduce the execution time and increase the write throughput that provides a better response time for the HDFS client. According to the experimental results, it is found that HDFS write operation throughput falls as file size rises. This is mainly due to the replication factor, the limitation of the network bandwidth and file block size. Therefore, the impact of file blocks size and replication factor on HDFS write performance have also examined.

The rest of this paper is organized as follows; related work is described in section II. The Anatomy of HDFS files writes pipeline and the lazy replication technique have been discussed in section III. Experimental results are presented and discussed in section IV. Finally, conclusions and future works are discussed in Section V.

II. Related Work

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

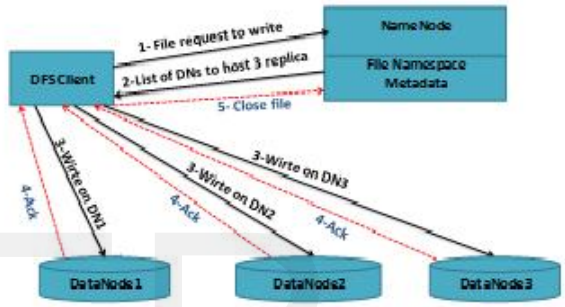

Fig.2. Writing a File on HDFS using Parallel (Broadcast)

Narendra M Patel .*et.al*. [13] has enforced the parallel manner in HDFS, where after requesting NameNode to write a file and receive a 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 the desired block in parallel. Once the block is written in DataNode2 and DataNode3, they send an acknowledgement to first DataNode. After getting acknowledgement from both DataNode2 and DataNode3, DataNode1 sends an acknowledgement to the client. Finally, the client sends an acknowledgement to NameNode that block is successfully written on three different nodes (see Fig. 3).

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 the DataNodes according to their past performance and tracks the information continuously. When a client initiates an upload request, the NameNode will send it a list of "high performance" DataNodes that it thinks will return the highest throughput for the client. By choosing higher performance DataNodes relative to each client and by taking advantage of the multi-pipeline design.

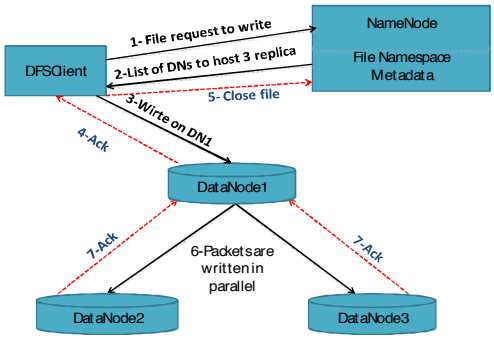

Fig.3. Writing a File on HDFS using Parallel (Master/Slave)

Eman.S.Abead . *et.al.* [15] has introduced a comparative study of the most common HDFS replication techniques; the default pipeline, parallel (Broadcast), and parallel (Master/Slave) technique. They provide the comprehensive and theoretical analysis of these existed HDFS replication techniques; The technical specification, features, and specialization for each technique along with its applications have been described.

III. THE ANATOMY OF HDFS FILE WRITE PIPELINE AND PROPOSED REPLICATION TECHNIQUE

An application adds data 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 ac knowledgement 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 st ored 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 fi nished 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 Lazy Replication Technique:

This section illustrates the components of the lazy replication technique. 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 writ ten on three different DataNodes; DataNode1, DataNode2 and DataNode3. The steps of writing a file in the HDFS according to the lazy technique are as follows:

1) A Client requests NameNode to write a file.
2) The Client first receives a li st of DataNodes to write and to host replicas of a single block.
3) The Client first writes a block to DataNode1.
4) Once a block is fi lled on DataNode1, DataNode1 sends an acknowledgement to the client. Then, the client sends an ackn owledgement to NameNode informing that the block is successfully written on one node; the client can request for next block to write.
5) DataNode1 creates a thread and request DataNode2 and DataNode3 to create replicas of the desired block in parallel.
6) Once, the block is written on DataNode2 and DataNode3; they send an acknowledgement to DataNode1.
7) 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. If DataNode1 fails to receive an acknowledgement from any of DataNode2 or DataNode3, it resends the same block to them.

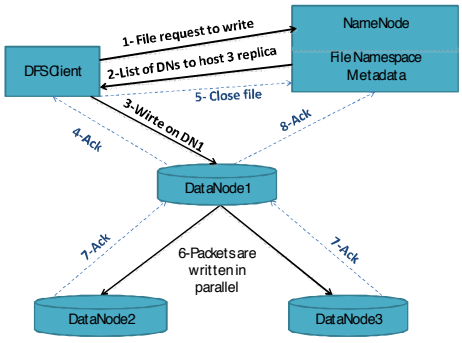

Fig.4. Writing a File on HDFS using Lazy Replication technique

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 as follows (see Fig. 5):

1) A Client requests NameNode to write a file.
2) The Client first receives a l ist of DataNodes to write and to host replicas of a single block.
3) The Client first writes a block to DataNode1 and DataNode2 in parallel.
4) Once a bloc k is fil led 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.
5) DataNode1 creates a thread and request DataNode3 to create replicas of the desired block in parallel.
6) Once, the block is w ritten on Dat aNode3; it sends an acknowledgement to DataNode1.
7) Finally, DataNode1 sends an ac knowledgement 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 ackn owledgement from DataNode3, it resends the same block to it.

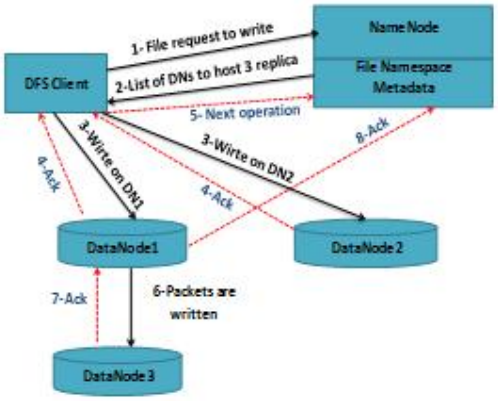

Fig.5. Writing a File on HDFS Using The Reconfigured Lazy Replication Technique

Another modification has been done 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 (see Fi g. 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 con figurable 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 D ataNode1 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 clie nt 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 ackn owledgement 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.

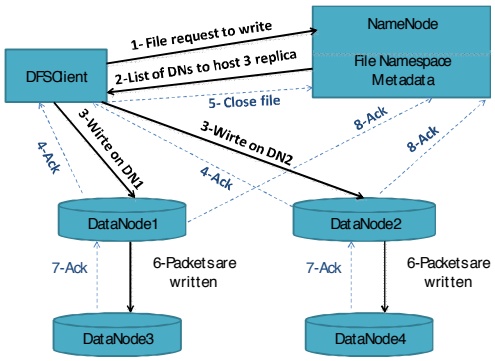

Fig.6. Writing a File on HDFS Using The Enhancement Lazy Replication Technique

IV. Performance Evaluation

In this section, the performance evaluation of the proposed Lazy, Lazy Reconfigurable, and enhancement Lazy techniques with respect to the pipeline, parallel (Broadcast) and parallel (Master/slave) 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 e ach NameNode/DataNodes. Also, the performance may vary as different cluster configuration environment varies.

A. Cluster Configurations.

The proposed 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 MapR educe 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 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, w rite and append operations. TestDFSIO is an appli cation available as part of the Hadoop distribution [19].

Fig 7(a,b) represents the expermintal results for HDFS file write with Replication Factor is three 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, 10% reduction of the parallel (Broadcast) replication technique, a nd 6% reduction of the parallel (Master/Slave) comparing to the pipelined replication technique.

Fig.7(b) represents the results of the throughput of the existed parallel (Broadcast), parallel(Master/Slave), Default pipelined and the lazy techniques According to the results it has observed that the throughput improvement is around 15% for the lazy HDFS replication technique, 10% for the parallel (Broadcast) replication technique, and 7% for the parallel (Master/Slave) 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 al so reduces the metadata size of t he 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.

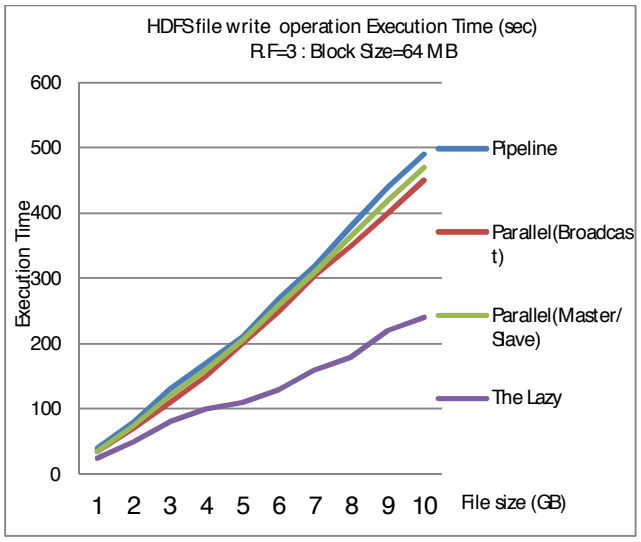

Fig. 7(a) TestDFSIO Execution Time (sec)

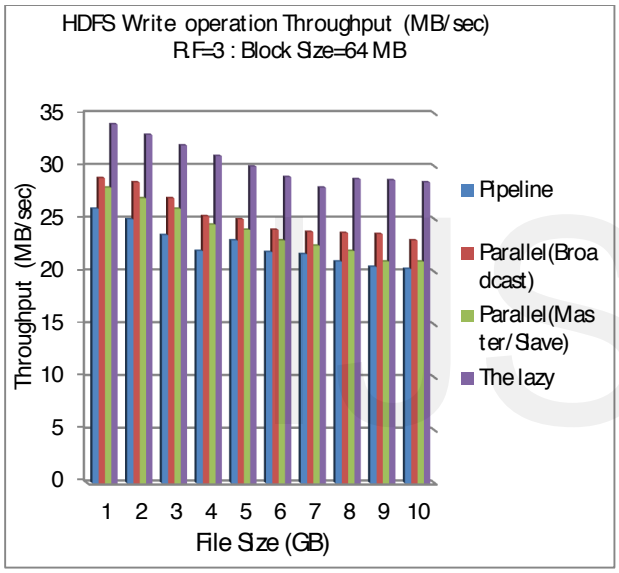

Fig. 7(b) TestDFSIO Throughput (MB/sec)

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 10% to 12% in parallel (Broadcast), and approximately 7% to 9% in parallel (Master/Slave) 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.

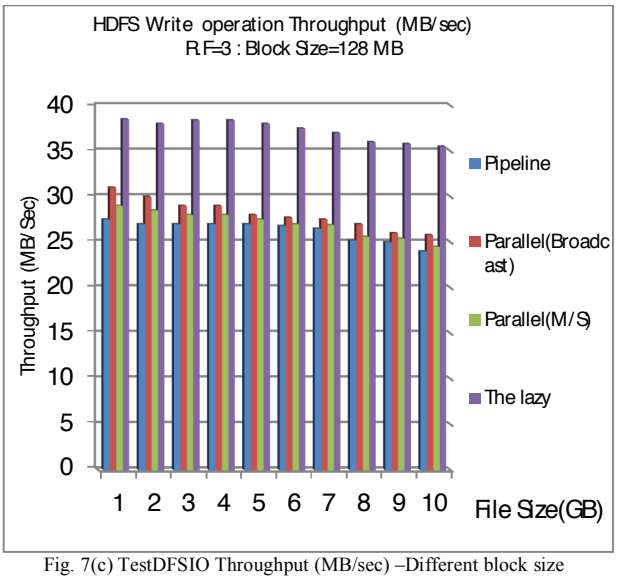

Fig. 7(c) TestDFSIO Throughput (MB/sec) –Different block size

The performance evaluation of the proposed reconfigurable lazt replication technique with respect to the default pipeline, parallel (Broadcast), and parallel (Master/ Slave) replications techniques is depicated in Fig. 8(a,b) for HDFS file write with replication factor is three 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.8(a), it is found that approximately 25% reduction in the execution time of the reconfigured lazy HDFS replication technique, 10% reduction of the parallel (Broadcast) replication technique, and 6% reduction of the parallel (Master/Slave) comparing to the pipelined replication technique

According to the results in Fig.8(b),. it has observed that the throughput improvement is around 12% for the reconfigured lazy HDFS replication technique, 10% for the parallel (Broadcast) replication technique, and 7% for the parallel (Master/Slave) 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 four techniques.

Fig.8(c) illustrates the performance of the four techniques by considering large block size. The improvement in file write throughput is approximately 12% to 15% in the reconfigured lazy technique, approximately 10% to 12% in parallel (Broadcast) and approximately 7% to 9% in parallel (Master/Slave) technique comparing to the pipeline technique.

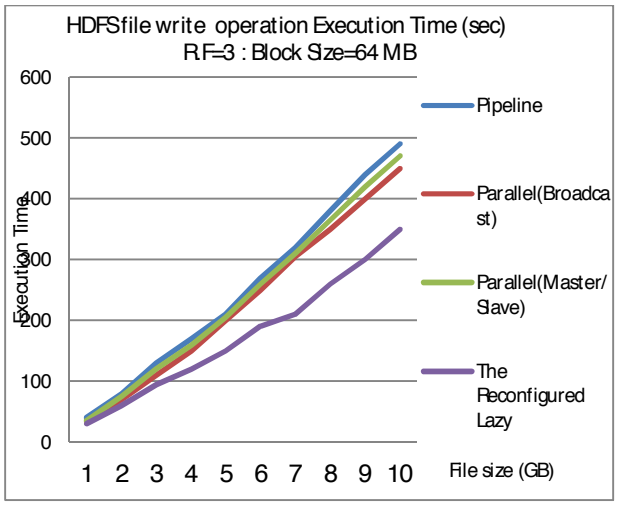

Fig. 8(a) TestDFSIO Execution Time (sec)

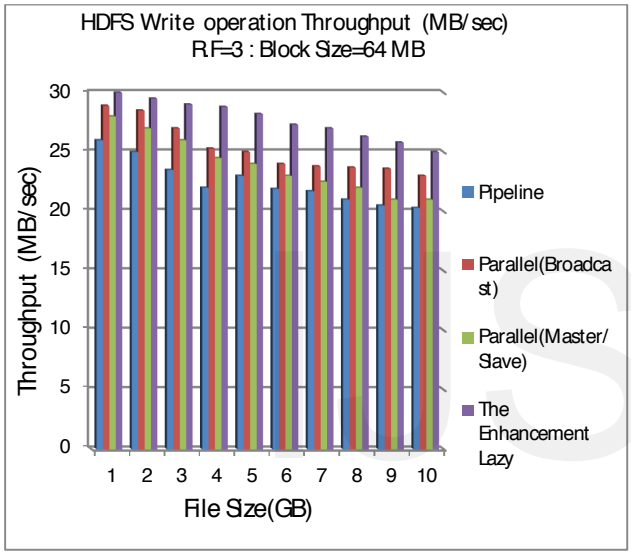

Fig. 8(b) TestDFSIO Throughput (MB/sec)

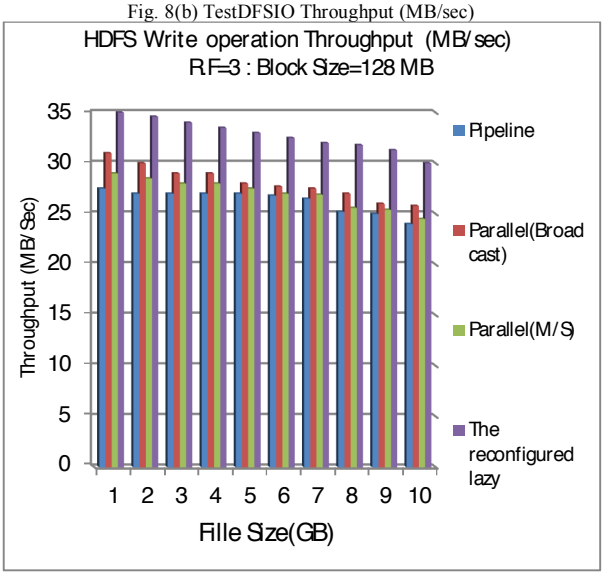

Fig. 8(c) TestDFSIO Throughput (MB/sec) –Different block size

V. Conclusions and Future Work

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 lazy replication has been introduced for efficient replica placement on HDFS that can increase write throughput. The lazy technique has been implemented using three DataNodes with two configurations

Another modification has been introduced to the proposed lazy technique by introducing extra DataNode as a backup for the DatNode1 to improve the availability without affecting the write throughput and the execution time.

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: 27/3/2015 1:00 AM). *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, "E rms: 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. Ga o, 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, 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 4-11

[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

[18] Ahmed M W ahdan 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.* Available: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/