A Comparative Study of HDFS Replication Approaches

Eman S.Abead
Faculty of Computers and Information
Cairo University, Egypt

Mohamed H. Khafagy
Faculty of Computers and Information
Fayoum University, Egypt

Fatma A. Omara
Faculty of Computers and Information
Cairo University, Egypt

Abstract— **The Hadoop Distributed File System (HDFS) is designed to store, analysis, transfers large scale of data sets, and stream it at high bandwidth to the user applications. It handles fault tolerance by using data replication, where each data block is replicated and stored in multiple DataNodes. Therefore, the HDFS supports reliability and availability. The data replication of the HDFS in Hadoop is implemented in a pipelined manner which takes much time for replication. Other approaches have been proposed to improve the performance of the data replication in THE Hadoop HDFS .The paper provides 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 describes the technical specification, features, and specialization for each approach along with its applications. A comparative study has been performed to evaluate the performance of these approaches using TestDFSIO benchmark. According to the experimental results it is found that the performance (i.e., the execution time and throughput) of the parallel (Broadcast) replication approach and the parallel (Master/Slave) outperform the default pipelined replication. Also, it is noticed that the throughput is decreased with increasing the file size in the three approaches.**

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

I. INTRODUCTION

Today, the data size of the used databases in the enterprises has been grown exponentially. On the other hands, the data is generated by many sources like business processes, transactions, social networking sites, web servers, etc. Also, the stored data might be structured, or unstructured form, or even both forms.

Now a day, the business applications features in the enterprises could be large-scale, data-intensive, and web-oriented that could be accessed from diverse devices including mobile devices. So, the processing, analyzing, and management of the huge amount of data are considered curtail and challenging problems [1].

Big Data concept means a dataset which grow continuously such that it becomes difficult to manage by using the existed database management concepts and tools.

The difficulty can be related to data capture, storage, search, sharing, analysis and visualization etc. Big Data spans across three dimensions; **V**olume, **V**elocity and **V**ariety [2].

- ***Volume*** the data becomes huge, and its size could be terabytes and petabytes.
- ***Velocity*** the needed data should be streamed to the enterprise in order to maximize its value with respect to the business. So, the time is considered very critical here.
- ***Variety*** the data should be structured .But it could include unstructured and semi-structured data. The unstructured data has different format such as text, audio, video, posts, and the semi-structured data might be log files like email etc.

The Apache Hadoop project develops open-source software for reliable, scalable and distributed computing. On the other hands, the Apache Hadoop

software library is a framework that allows to process large data sets which are distributed across clusters of computers using a simple programming model. It enables the applications to be executed using thousands of computational independent computers and petabytes of data. Hadoop has derived from Google's MapReduce and Google File System (GFS) **[2, 3].**

MapReduce Programming Framework:

MapReduce is a software framework which has been introduced by Google in 2004 to support the distributed computing on large data sets using clusters of computers [4]. The main components of the MapReduce are Map, and Reduce. The function of the Map component is to process a key/value pair to generate a set of intermediate key/value pairs. The function of the Reduce component is to merge all intermediate values associated with the same intermediate.

Hadoop Distributed File System (HDFS):

The Hadoop Distributed File System (HDFS) is a file system which is designed for large-scale distributed data processing under frameworks such as MapReduce [5]. HDFS supports fault tolerance and it is designed to run on commodity hardware. It provides high throughput access to the application data. On the other hands, the HDFS can store data across thousands of servers, and process the Hadoop functions (i.e., Map/Reduce functions) across these machines such that the data is always available when it is needed. The HDFS has master/slave architecture, and large data is automatically split into chunks which are managed by different nodes in the Hadoop cluster [6]. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node, in the cluster, which manage storage attached 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 [7]. When The HDFS client opens a file for writing, the NameNode allocates a block with a unique block ID and determines a list of DataNodes to host replicas of that block. The DataNodes form a pipeline. Client writes data block on first DataNode then data are pushed to the next DataNode in pipeline manner. Acknowledgement of data written on DataNodes is also received in pipeline. After all the replicas are written correctly, the client request NameNode to write the next block. These kinds of pipelined replication scheme affect the performance of file write operation [8]. Fig.1 represents HDFS architecture.

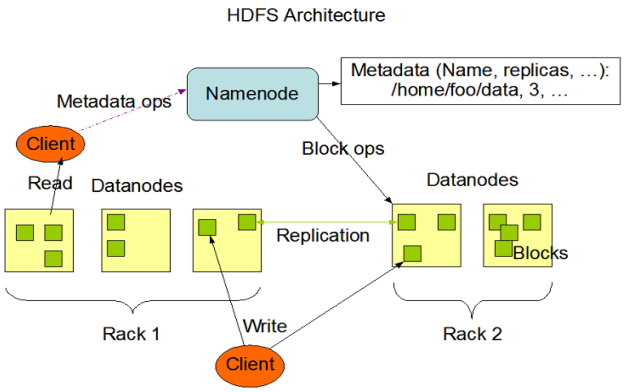

Fig. 1 HDFS Architecture

The challenges issues of the Data Replication in Hadoop include timing, scalability, availability, connection overhead and fault tolerance. These issues can be related to reducing the total wait time for acknowledgements in the HDFS Client side.

In this paper, we compare HDFS replication approaches from three different research, the default pipeline approach, parallel (Broadcast) approach and parallel (Master/Slave) approach. Data Replication in Hadoop challenging issues includes the Timing, Scalability, Availability, Reducing connection creation overhead in HDFS Client and fault tolerance. These issues can be related to reducing the total wait time for acknowledgements in the HDFS Client side. Experimental results of all approaches tested by using the TestDFSIO benchmark which gives better response time to HDFS client.

The rest of this paper is organized as follows . The default pipeline approach, parallel (Broadcast) approach and parallel (Master/Slave) approach is described in section II. The comparative study and Experimental results were shown III. Finally, conclusions of the paper in Section IV.

II. THE HDFS FILE WRITE PIPELINE AND REPLICATION APPROACHES

The placement of data replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization [7].

The default strategy of Hadoop HDFS is that the first replica is placed on the same node as the client (i.e., clients running outside the cluster, a node is chosen at random, although the system tries not to pick nodes that are too full or too busy). The second replica is placed in different rack from the first (off-rack), which is chosen randomly. The third replica is placed on the same rack as the second, but on a different node which are chosen randomly. Further replicas are placed on random nodes in the cluster, where the system tries to avoid placing too many replicas on the same rack [9].

For the sake of the comparative study, the main principles of the three existed HDFS replication approaches; default pipeline, parallel (Broadcast), and parallel (Master/Slave) approaches will be discussed in details.

A. Pipeline Replication Approach:

The HDFS is designed to reliably store very large files across machines in a 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 a file are replicated for fault tolerance. The block size and the replication factor are configurable per file. An application can specify the number of replicas of a file at the file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The steps of the default pipeline replication approach are (see Fig.2) **[6].**

1) HDFS client sends a request to the NameNode to create a new file in the filesystem's namespace.
2) NameNode returns list of DataNodes to store data block according to the 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. By considering the replication factor is three, so there are three nodes in the pipeline.
4) The packets are sent to the DataNode1 in the pipeline, which stores the packet and forwards it 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 the DataNodes also receives 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 the DataNode pipeline and waits for acknowledgments before contacting the NameNode to signal that the file is complete.

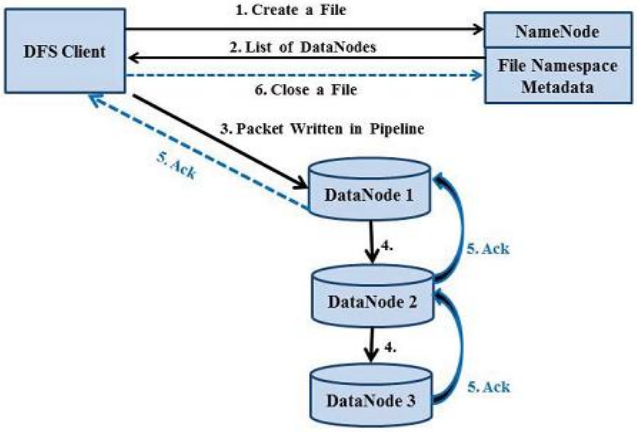

FIG.2 WRITING A FILE ON HDFS USING PIPELINE REPLICATION APPROACH

B. Parallel(Broadcast) Replication Approach:

According to the parallel approach of the HDFS, The client writes all replicas in parallel in the DataNodes. This approach improves the write performance. In order to perform parallel replication, two new classes are used; Multi DataOutputStream class and MultiDataInputStream class. The steps of the parallel replication approach are (see Fig.3) [8]:

1) Client sends a request to the NameNode to create a new file in the file system's namespace.
2) NameNode returns list of DataNodes to store data block according to the replication factor
3) HDFS client's file data are first divided into default block size then splits into packets

which encapsulate multiple output streams into one object for parallel writing.

4) In this case, packets are written to all three DataNodes instead of single DataNode.
5) HDFS client receives Acknowledgements from all DataNodes instead of just the first one as in the pipeline. MultiDataInputStream class reads acknowledgement from MultiDataInputStream instance.
6) When the client has finished writing data, it calls *close()* on the stream.

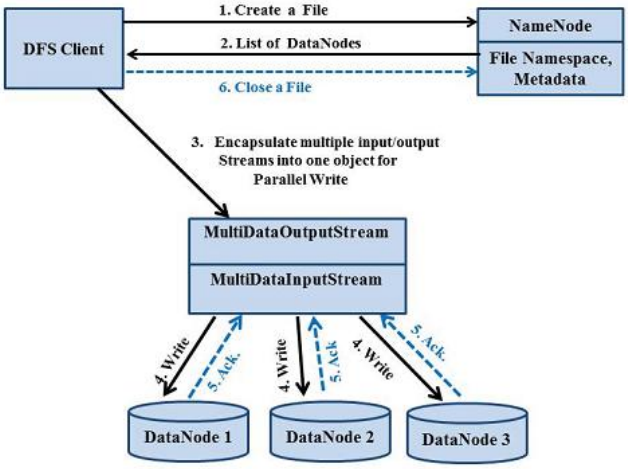

FIG.3 WRITING A FILE ON HDFS USING PARALLEL (BROADCAST) REPLICATION APPROACH

C. **Parallel (Master/Slave) Replication Approach:**

According to this approach, the creation and writing of a file are faster than that the Hadoop DFS file. NameNode (NN) never writes any data directly on DataNodes, but it manages the namespace and node only. In this approach, a single block is written in three different DataNodes, Assume DN1, DN2 and DN3. The steps of the parallel Master/Slave approach are (see Fig.4) [10].

1) Client sends a request to the NameNode to write a file.
2) The Client receives the list of DataNodes to write and to host replicas of a single block.
3) Client first writes a block to DN1.
4) Once a block is filled in DN1, DN1 creates thread and requests DN2 and DN3 for creating replicas of a desired block in parallel.
5) Once the first block is written in DN2 and DN3, they send an acknowledgement to DN1.
6) After getting acknowledgement from both DN2 and DN3, DN1 sends acknowledgement to client. If DN1 fails to receive acknowledgement from any of DN2 or DN3, it resend the same block again to them.
7) Finally, the client sends acknowledgement to NameNode that block is successfully written on three different nodes.

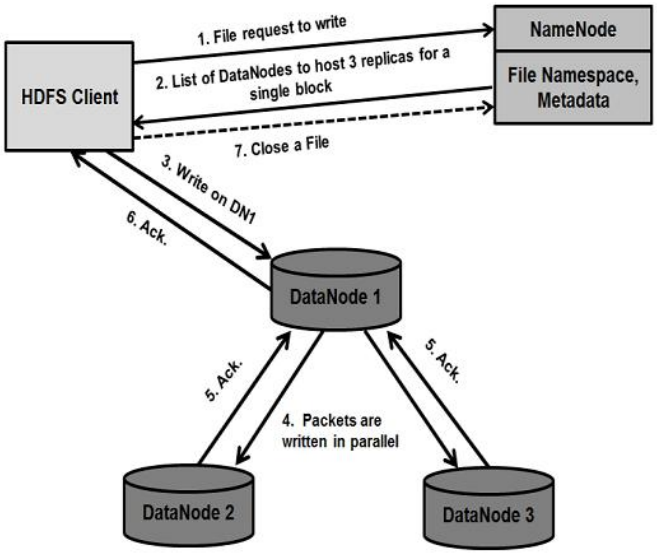

FIG.4. WRITING A FILE ON HDFS USING PARALLEL (MASTER/SLAVE) REPLICATION APPROACH

III. COMPARATIVE STUDY AND EXPERIMENTAL SETUP

In this section, the performance evaluation of the three HDFS replication approaches 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 and DataNodes. Also, the performance may vary as different cluster configuration environment varies.

A. Cluster Configurations:

The three HDFS replication approaches are 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, 3.2 GHz processor with 16GB RAM and 320GB SATA HDD. Each node runs Ubuntu 14.10 In all experiments, Hadoop framework 1.2.1 and JDK 1.7.0 are used. These nodes locate in three different racks with Gigabit Ethernet network connecting.

B. **The Performance Evaluation**:

The three replication approaches have been implemented and tested using TestDFSIO benchmark to evaluate their impact on the HDFS write throughput. The TestDFSIO benchmark is a read and write 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 a cluster machines (particularly the NameNode and the DataNodes) . TestDFSIO measures the average throughput for read, write and append operations. TestDFSIO is an application available as part of the Hadoop distribution [11].

According to the experiment results in Fig.5(a), it is found that approximately 10% reduction in the execution time for the parallel (Broadcast) replication approach and 6% reduction for the parallel (Master/Slave) comparing to the pipelined replication approach for HDFS file write with Replication Factor is three and Block Size is 64MB.

According to the experiment results in Fig.5(b), It has observed that the throughput improvement is around 10% for the parallel (Broadcast) replication approach , and 7% for the parallel (Master/Slave) 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 three approaches.

Fig.5(b) and Fig.5(c) illustrate the performance of the three approaches by considering large block size. The improvement in file write throughput is approximately 10% to 12% in parallel (Broadcast) and pipeline approaches, and approximately 7% to 9% in parallel (Master/Slave) and pipeline approaches.

The Replication factor and the limitations of the network bandwidth are also affected the file write throughput. To study the effects of the Replication factor, the experiments have been implemented with considering the replication factor of a file three and two. The experimental results show that the improvement of write throughput for parallel (Broadcast) approach and pipeline approach is up to 20% , and up to 10% for parallel (Master/Slave) and pipeline approaches (see Fig.5(d)). However, for a small number of replicas, there is only slightly improvement has been provided by the parallel replication approaches.

Also, the file size and its blocks size would be affected the performance of the HDFS writing. A file with small number of blocks (i.e., blocks size is large) will potentially make the client to read/write more data without connecting with the NameNode, and it also reduces the metadata size of the NameNode, and reduces NameNode workload. This would be important for large file systems. Hence, by larger file size, and larger number of blocks, the total number of the requests from the HDFS clients to NameNode will be increased which leads to increase the network overhead.

The experiments have been implemented with considering the block size is 128 Mbytes instead of 64 Kbytes. The experimental results show that the performance of throughput of the three approaches has been affected.

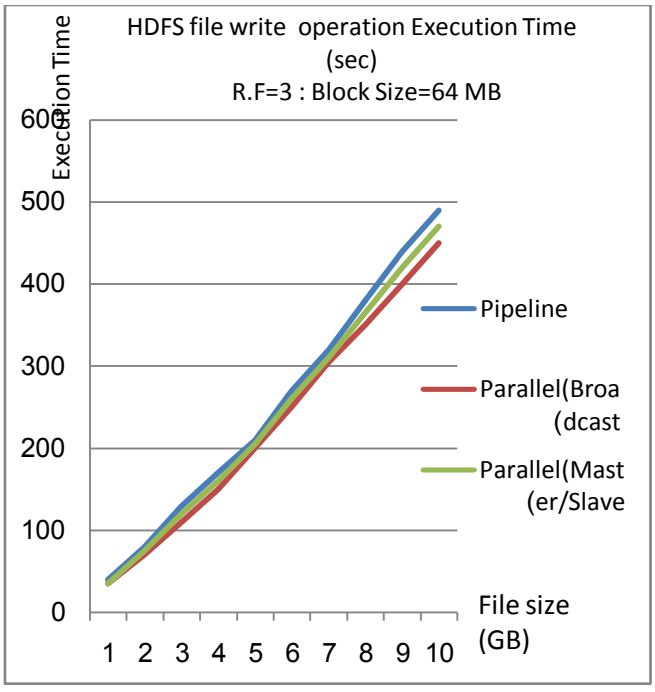

Fig. 5(a) TestDFSIO Execution Time (sec)

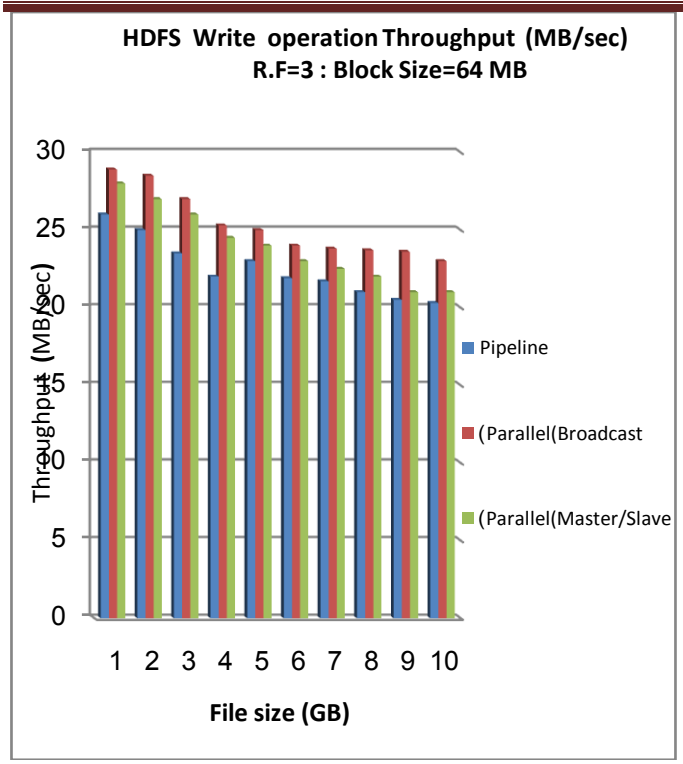

Fig. 5(b) TestDFSIO Throughput (MB/sec)

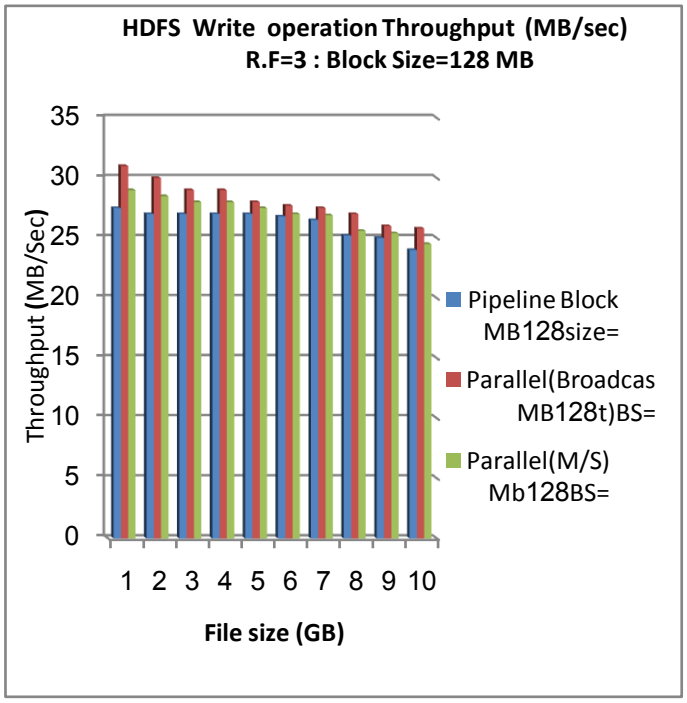

Fig. 5(c) TestDFSIO Throughput (MB/sec) - Different block size

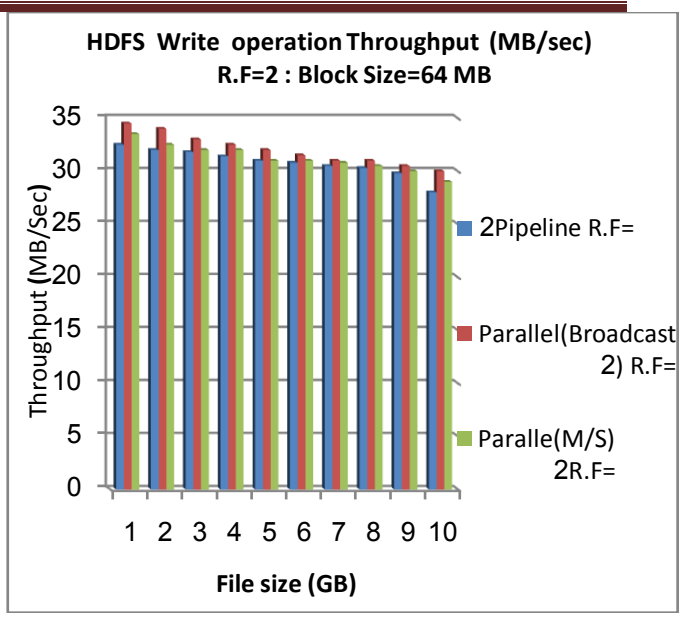

Fig. 5(d) TestDFSIO Throughput (MB/sec) - Different R.F

IV. CONCLUSION

Data replication is a technique commonly used to improve data availability. In HDFS each block is replicated and stored in different nodes.

In this paper, a comparative study using the TestDFSIO benchmark has been done to evaluate the performance of three existed HDFS replication approaches; the default pipeline approach, parallel (Broadcast) approach and parallel (Master/Slave) approach. According to the experimental results it is found that the execution time of the parallel (Broadcast) replication approach and the parallel (Master/Slave) are improved by approximately 10% and 6% comparing to the pipelined replication approach by considering the Replication Factor is three and the Block Size is 64 MB. Also, the throughput of the parallel (Broadcast) replication and the parallel (Master/Slave) replication approaches has been improved by 10%, and 7% respectively comparing to the default pipelined replication. From the results, it is noticed that the throughput is decreased with increasing the file size in the three approaches.

Generally, for efficient replica placement of the HDFS to improve write throughput and execution time, different HDFS parameters like file size, block size, and Replication Factor should be considered.

REFERENCES

[1] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, 2012, pp. 1-5.

[2] P. Russom, "Big data analytics," *TDWI Best Practices Report, Fourth Quarter,* 2011.

[3] (Access : 27/3/2015 1:00 AM). *HDFS Architecture*
Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.

[5] C. Lam, *Hadoop in action*: Manning Publications Co., 2010.

[6] 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.

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

[8] 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 Conferences*, 2014, pp. 36-39.

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

[10] 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.

[11] 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/