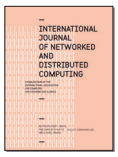
International Journal of Networked and Distributed Computing
Vol. **8**(**3**); June (2020), *pp.* 119–123
DOI: https://doi.org/10.2991/ijndc.k.200515.007; ISSN 2211-7938; eISSN 2211-7946
https://www.atlantis-press.com/journals/ijndc



Research Article

High Performance Hadoop Distributed File System

Mohamed Elkawkagy*, Heba Elbeh

Computer Science Department, Menoufia University, Shibin Elkom, Menoufia, Egypt

ARTICLE INFO

Article History

Received 13 March 2020
Accepted 24 April 2020

Keywords

Cloud
HDFS
fault tolerance
reliability

ABSTRACT

Although by the end of 2020, most of companies will be running 1000 node Hadoop in the system, the Hadoop implementation is still accompanied by many challenges like security, fault tolerance, flexibility. Hadoop is a software paradigm that handles big data, and it has a distributed file systems so-called Hadoop Distributed File System (HDFS). HDFS has the ability to handle fault tolerance using data replication technique. It works by repeating the data in multiple DataNodes which means the reliability and availability are achieved. Although data replications technique works well, but still waste much more time because it uses single pipelined paradigm. The proposed approach improves the performance of HDFS by using multiple pipelines in transferring data blocks instead of single pipeline. In addition, each DataNode will update its reliability value after each round and send this updated data to the NameNode. The NameNode will sort the DataNodes according to the reliability value. When the client submits request to upload data block, the NameNode will reply by a list of high reliability DataNodes that will achieve high performance. The proposed approach is fully implemented and the experimental results show that it improves the performance of HDFS write operations.

1. INTRODUCTION

A Distributed File System (DFS) is client/server software paradigm. It gives the client's ability to access the server data. In general, DFS includes software exist on the servers and clients that connects files on different file servers into a single Namespace to improve the data availability. In case of a client retrieving a file from the server, the retrieved file will appear on the client machine as a normal file, so the client able to access the file as if it were stored locally on the client machine. After that, client stores the modified file to the server.

As shown in Table 1, there are many types of DFSs such as Network File System (NFS). Sandberg et al. [1] introduced NFS protocol. Sun Microsoft laboratory published the NFS protocol specification as a public use by other vendors. After that Howard [2] presented the Andrew File System (AFS). It was inspired by Carnegie Mellon University. It designed for working with distributed workstation environment. The Google File System (GFS) was inspired by Ghemawat et al. [3]. GFS has many features as previous DFSs such as performance, scalability, reliability, and availability. GFS cluster includes a single master and many segment servers and can be invoked by more than one client. With the appearing object oriented in the computing environment, a file system so-called XtreemFS was developed by Martini et al. [4]. It was designed for Grid environments.

The main essence of the proposed approach is improving the Hadoop write operation by using multi-pipeline technique instead of single pipeline and improving the quality of DataNodes by adapting its reliability to deal with the best DataNodes and ruling out the DataNodes that has low reliability.

The rest of this paper is organized as follows. The Hadoop DFS (HDFS) operations are discussed in Section 2. After that the proposed approach is introduced in Section 3. In Section 4, the experimental results is introduced. Finally, conclusion is summarized in Section 5.

2. HADOOP DISTRIBUTED FILE SYSTEM

To design a DFS you have to use one of the architecture types; Client–Server, Parallel, Centralize or Decentralize architecture. Hadoop DFS uses Client–Server architecture which includes only one NameNode and a lot of DataNodes. The different between HDFS and other DFS are significant. The HDFS is designed to accomplish high rate either in data transfer or throughput.

An HDFS [5,6] includes a single master node so-called NameNode. The responsibility of this node is to manage the file system namespace and organizes client requests such as invoking files. Also, it includes a set of DataNodes, usually one node for each cluster. These nodes handle the attached storage. In general, the file is divided into a number of blocks to store in DataNodes. The NameNode performs the file system namespace operations such as opening, and closing files as well as mapping the blocks to the DataNodes. The DataNodes able to perform the client operations and can create, delete the blocks, and replication upon instruction that comes from the NameNode. Generally, both nodes either data

*Corresponding author. Email: M_nabil_shams@yahoo.com

Table 1 | DFS types

File system features	NFS	AFS	GPS	XFS	HDFS
Reliability	NA	√	√	√	√
Flexibility	NA	Limited	√	√	√
Scalability	Limited	√	√	√	√
Transparency	NA	Limited	√	√	√
Security	√	√	√	Limited	Limited
Fault tolerance	Limited	Limited	√	√	√ (Waste much time)

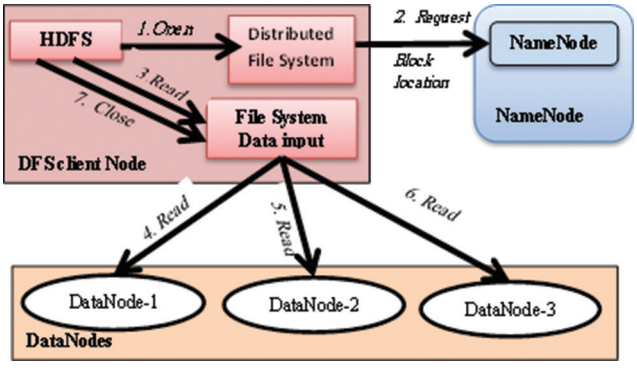


Figure 1 | HDFS read operation.

or name nodes manage the clients request either to read/write to/from the file system clients.

2.1. HDFS Read Operation

In order to read from HDFS, firstly the client should communicate with the NameNode. NameNode includes all the data about DataNodes (metadata). Therefore, the client will communicate the determined DataNodes to read the required data. As discussed in Sivaraman et al. [7], to achieve the security/authentication feature, NameNode uses token to the client which clarify that the client read data from the DataNode.

As depicted in Figure 1, the read operation in the Hadoop HDFS is organized as follows:

(1) Through the distributed file system Application Programming Interface (API), the client asks NameNode to send the location of required block.

(2) Namenode checks the privilege of the client to access the required data.

(3) If the client is an authorized user, the NameNode reply by the data location and security token for authentication purpose. Otherwise, the NameNode reject the client request.

(4) At the DataNode side, after checking the security token, DataNode gives the client permission to read that required block.

(5) So, the client starts to read data directly from DataNode through opening an input stream.

(6) In case of the file DataNode breaks down, and then the client will return to NameNode to get other location for the same block.

2.2. HDFS Write Operation

In the writing operation, the client contact with the NameNode as in reading process to get the address of the DataNode which stores the data [7].

As shown in Figure 2, the writing operation in the Hadoop HDFS is organized as follows:

(1) The client contact with the NameNode through the distributed file system API to get the location of the DataNode which store the data.

(2) Through FS data output stream, the client begins writing the data.

(3) Once the client finishing writing the first block, the first DataNode will replicate the same block to other dataNode. And then this DataNode, starts copying this block to the next DataNode after receiving the block.

After storing and replicating the data, DataNode sends an acknowledgment to the client to inform it the data is stored successfully.

Due to the failures which can occur in hardware or software in DFSs, these distributed files should apply fault-tolerance technique to handle the faults. HDFS is highly fault tolerant. It uses replica process to handle faults. This means client data is repeated many times (default replica factor is 3) on different DataNode in the HDFS cluster. So that in case of any DataNode goes down, the data will be accessed from other DataNodes. In general, there are a lot of techniques that have been introduced to handle fault tolerance in HDFS.

Based on the way of accessing data blocks during runtime, Abad et al. [8] proposed a dynamic data replication scheme to improve the locality of data with the data replication in HDFS. After that, Fan et al. [9] proposed updated replica process in HDFS to handle asynchronous encoding and overcome the redundancy overhead in DARA system [8]. Cheng et al. [10] introduced an elastic and dynamic replica management system. The objective of this approach is increasing the locality of hot data more than the cold data. In addition, it can improve the network overhead and enhance the data reliability and availability.

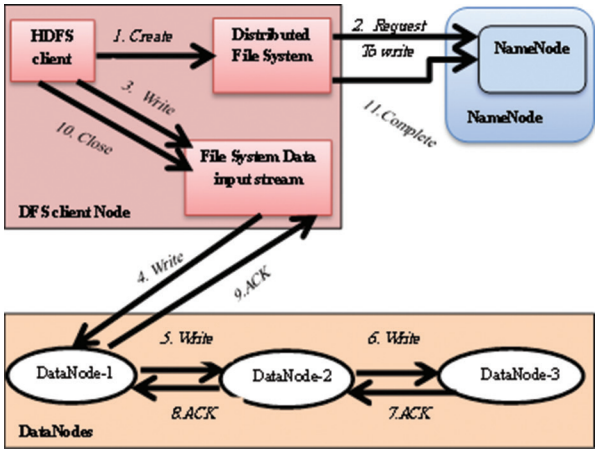


Figure 2 | HDFS write operation.

Feng et al. [11] proposed an approach so-called Magicube. This architecture simultaneously achieves high reliable and low redundancy because it uses one replica in the HDFS, and (n, k) algorithm for fault tolerance. This approach works very well with the batch system because the fault tolerance algorithm is executed in the background.

Patel Neha et al. [12] introduced parallel technique to improve the replica placement in HDFS throughput. Through this technique, the client can upload a lot of data without referring to the NameNode, and can reduce the metadata size and loading in the NameNode. After that, Patel Neha et al. [13] improved the throughput of the HDFS by replicating blocks of data in parallel and transferring a large size of data blocks which increase rate of data transfer.

SMART Hadoop file system (SMARTH) approach was introduced by Zhang et al. [14]. Instead of using stop and wait technique with single pipeline protocol, SMARTH approach used synchronous multi-pipeline protocol with a revised fault tolerance mechanism.

Kashkouli et al. [15] introduced investigation for the Hadoop architecture in distributed system and hence compared the Hadoop with Map-Reduce.

Singh and Singh [16] proposed a new approach to detect the nodes that has fault and hence rule out them to improve the performance of job execution of the cluster.

Poola et al. [17] discussed and introduced a depth taxonomy of fault-tolerant mechanisms in different distributed systems. In addition, they provide ontology for the faults and fault-tolerant techniques.

3. THE PROPOSED APPROACH

In this section, we will introduce the proposed technique and show how the proposed design will consider the reliability value to decide the best DataNodes as well as considering the full network bandwidth when writing data into an HDFS cluster.

As depicted in Figure 3, the proposed technique works as follows:

(1) HDFS client asks NameNode to construct a new file in the file system's namespace.

(2) According to the reliability of each DataNodes, NameNode replies by ordered list of DataNodes to store data. (Number of DataNode in the list is determined according to the repetition times. The default value is 3).

(3) HDFS client's file data is divided into blocks with default size, and then divide the block into packets. The number of DataNodes forms a pipeline. According to the default replication, there are three nodes in the pipeline.

(4) The packets are sent to the first DataNode and the second DataNode in parallel. In the same way, the second DataNode stores the packet and then forwards it to the third DataNode in the pipeline.

(5) Then the first and second DataNode send acknowledge to the clients. Then client sends acknowledge to the NameNode informing that the block is successfully written on two nodes. The client will request for the next writing operation without waiting the other acknowledge of the NameNode. This means, Additional pipelines will be created for sending new block.

(6) At the same time, each DataNode will send an update message to the NameNode to update its reliability value according to reliability rating algorithm. For each DataNode the reliability evaluation algorithm is applied. The algorithm starts with initializing the reliability for each DataNode by 1. The reliability value is adapted through the variable *Adapt-F* which has value greater than 0 (lines 1–3). The inputs of the algorithm are *F-repli, minReliability* and *maxReliability* (line 3). *F-repli* represents reliability factor which controls the rate of increasing or decreasing the reliability of the DataNode. *MinReliab* and *MaxRelib* indicate the minimum and maximum reliability level respectively. In case of the DataNode, that exceeds *MaxReliab* value, it will take the *MaxReliab* value because reliability value should be compatible with the lower reliability to easiest converges towards lower reliability (lines 11–12). On the other side, the DataNode that reach to *MinReliab* or less than *MinReliab*, is stoped and call the repairing procedure (lines 13–15).

Note that, the multiplication with the adaptability factor *Adapt-F* in case of failing helps the system to decrease the reliability value of the DataNode more quickly than increasing the reliability value in case of success (lines 4–10).

(7) Once the packet stored in the DataNode3, it sends acknowledge to the DataNode2 and at the same time update its reliability value in the Name node according to the equations in step number 6.

To this end, the NameNode should maintain table consists of three columns: Rack name, DataNode name, and the reliability value.

4. EVALUATION

A common evaluation of HDFS in a large cluster is to utilize its DataNodes from different racks. In general, in the same rack, the network bandwidth between DataNodes is larger than the network bandwidth between DataNodes in different racks.

The traditional HDFS strategy as follows, the NameNode will select the not busy DataNodes to store the data block. Then it will store the second replica in DataNode from different rack, while third one is placed in different DataNode on the same rack as the second. However, the performance of the traditional strategy is costly.

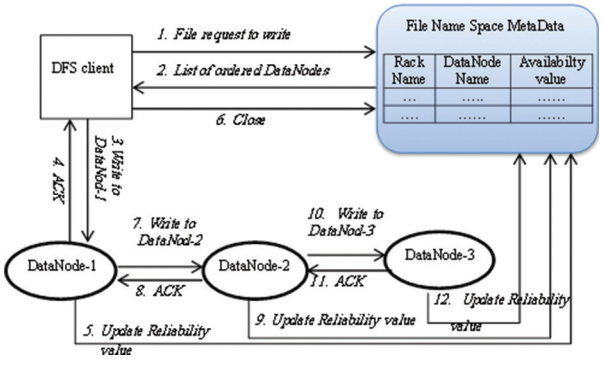


Figure 3 | Workflow of the proposed technique.

Algorithm: Reliability Assessment

1. **Begin**
2. *Initially* reliability:=1, Adapt-F:= 0.01
3. **Input** F-repli, maxReliability, minReliability, DataNodeStatus
4. **if** DataNodeStatus =success **then**
5. reliability:= reliability + (reliability * F-repli)
6. **if** Adapt-F > 1 **then**
7. Adapt-F:= Adapt-F –1;
8. **else if** DataNodeStatus = Fail **then**
9. reliability:= reliability – (reliability * F-repli * Adapt-F)
10. Adapt-F:= Adapt-F +1;
11. **if** reliability >= maxReliability **then**
12. reliability:= maxReliability
13. **if** reliability < minReliability **then**
14. DataNodeStatus:=Idle
15. call_proc: repair DataNodeStatus
16. **End**

To evaluate the effectiveness of the proposed writing on HDFS approach, we used the TestDFSIO benchmark. This benchmark is used to test HDFS operations either read or write. It is helpful to measure the performance of network, OS and Hadoop setup in both sides: NameNode and the DataNodes. In addition, it helps us to measures the average throughput for HDFS operations. TestDFSIO is an application available as part of the Hadoop distribution [18].

As shown in Table 2, our experimental was done with three Replication factor, block size 64 and 128 MB, file size ranges from 1 to 10 GB. In addition, the initial value for the reliability of each DataNode is 1, adaptability factor is 0.01, and maxReliability and minReliabilty are 1.4 and 0.06 respectively.

Figure 4 shows the execution time evaluation of the proposed approach for HDFS writing operation with block size 64 MB.

Table 2 | Parameters of the proposed approach

Experimental factor	Value
Replication factor	3
Block size	64 and 128 MB
File size	1:10 GB
Reliability	1
Adapt-F	0.01
maxReliability	1.4
minReliability	0.06

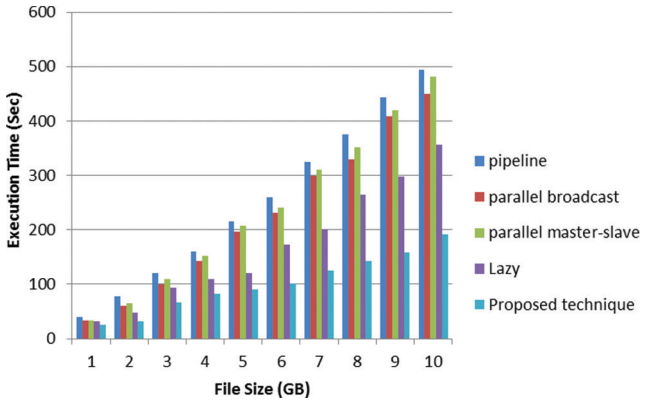


Figure 4 | HDFS file write operation execution time - Block size = 64 MB.

The experimental shows that there is no big gain when the size of the file is small. The proposed approach achieves an improvement of 35.9% when compared with traditional HDFS with file size 1 GB and reach to 61.3% with file size 10 GB. The overall reduction in the execution time of the proposed approach is reached. The experimental shows that there is no big gain when the size of the file is small. The proposed approach achieves an improvement of 35.9% when compared with traditional HDFS with file size 1 GB and reach to 61.3% with file size 10 GB. The overall reduction in the execution time of the proposed approach is reached to 48.9% when compared by parallel broadcast algorithm, and finally, outperform parallel master-slave technique by 60.3% in large file size. Figure 4 shows that the execution time takes 208 s to upload a 5 GB file in parallel master-slave technique, but the proposed approach only takes 91 s, which is 56.25% faster.

Figure 5 shows the execution time evaluation of the proposed approach for HDFS writing operation with block size 128 MB.

The proposed approach achieves an improvement of 22.8% when compared with lazy HDFS with file size 1 GB and reach to 34.03% with file size 10 GB. The overall improvement in the execution time of the proposed approach is reached to 65.06% when compared by traditional HDFS.

In general, when writing a file to HDFS using proposed approach, the time consumed is proportional to the file size either with the 46 or 128 GB.

Figures 6 and 7 show the improvement of the throughput for different techniques with considering 64 and 128 block size.

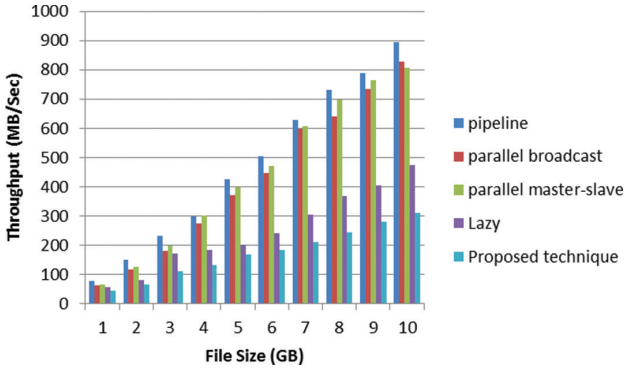


Figure 5 | HDFS file write operation execution time - Block size = 128 MB.

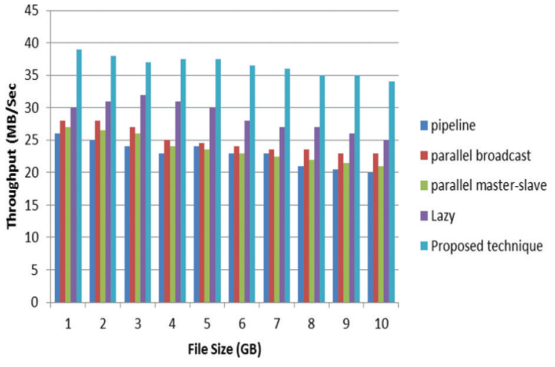


Figure 6 | HDFS file write operation Throughput - Block size = 64 MB.

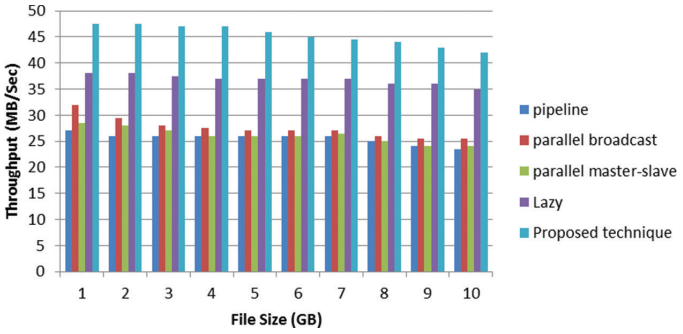


Figure 7 | HDFS file write operation Throughput - Block size = 128 MB.

The throughput improvement in the writing operation with 64 MB block size is ranged from 28.2% to 32.4% in the parallel broadcast technique, and 33.33% to 41.4% in traditional HDFS technique comparing to the proposed technique. On the other hand, in overall, the percentage of throughput improvement with 128 MB block size reach to 18.7% in lazy technique.

5. CONCLUSION

In the traditional HDFS, a data replication technique is used to accomplish data reliability and availability. It works by replicating data block on different DataNodes, which selected randomly; and uses single pipeline technique to send the data blocks. In this paper, we introduced a new approach to improve the replication technique by selecting the best DataNodes according to the reliability value. In addition, our approach uses parallel pipelined architecture to replicate the data blocks which increase the rate of data transfer and of course increase the system throughput by 59.5% with block size 64 MB and 43.6% with file size 128 MB.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

REFERENCES

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, Design and Implementation of the Sun Network Filesystem, Proceedings of the USENIX Conference & Exibition, Portland, OR, 1985, pp. 119–130.

[2] J.H. Howard, An overview of the Andrew file system, Proceedings of the USENIX Winter Technical Conference, Dallas TX, 1988, pp. 23–26.

[3] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, SOSP '03 Proceedings of the 19th ACM Symposium on Operating Systems Review, ACM, New York, USA, 2003, pp. 29–43.

[4] B. Martini, K.K.R. Choo, Distributed filesystem forensics: XtreemFS as a case study, Dig. Invest. 11 (2014), 295–313.

[5] W. Lin, L. Chen, B. Liu, A Hadoop-based efficient economic cloud storage system, 2011 Third Pacific-Asia Conference on Circuits, Communications and System (PACCS), IEEE, Wuhan, China, 2011, pp. 1–4.

[6] E. AbdElfattah, M. Elkawkagy, A. El-Sisi, A reactive fault tolerance approach for cloud computing, 2017 13th International Computer Engineering Conference (ICENCO), IEEE, Cairo, Egypt, 2017, pp. 190–194.

[7] E. Sivaraman, R. Manickachezian, High performance and fault tolerant distributed file system for big data storage and processing using hadoop, 2014 International Conference on Intelligent Computing Applications (ICICA), IEEE, Coimbatore, India, 2014, pp. 32–36.

[8] C.L. Abad, Y. Lu, R.H. Campbell, DARE: adaptive data replication for efficient cluster scheduling, 2011 IEEE International Conference on Cluster Computing, IEEE, Austin, TX, USA, 2011, pp. 159–168.

[9] B. Fan, W. Tantisiriroj, L. Xiao, G. Gibson, DiskReduce: RAID for data-intensive scalable computing, Proceedings of the 4th Annual Workshop on Petascale Data Storage, ACM, Portland, OR, USA, 2009, pp. 6–10.

[10] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, et al., ERMS: an elastic replication management system for HDFS, 2012 IEEE International Conference on Cluster Computing Workshops, IEEE, Beijing, China, 2012, pp. 32–40.

[11] Q. Feng, J. Han, Y. Gao, D. Meng, Magicube: high reliability and low redundancy storage architecture for cloud computing, 2012 IEEE Seventh International Conference on Networking, Architecture and Storage (NAS), IEEE, Xiamen, Fujian, China, 2012, pp. 89–93.

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

[13] M. Patel Neha, M. Patel Narendra, M.I. Hasan, M. Patel Mayur, Improving data transfer rate and throughput of HDFS using efficient replica placement, Int. J. Comput. Appl. 86 (2014), 254–261.

[14] H. Zhang, L. Wang, H. Huang, SMARTH: enabling multi-pipeline data transfer in HDFS, 2014 43rd International Conference on Parallel Processing, IEEE, Minneapolis MN, USA, 2014, pp. 30–39.

[15] A. Kashkouli, B. Soleimani, M. Rahbari, Investigating Hadoop architecture and fault tolerance in map-reduce, Int. J. Comput. Sci. Netw. Secur. 17 (2017), 81–87.

[16] C. Singh, R. Singh, Enhancing performance and fault tolerance of Hadoop cluster, Int. Res. J. Eng. Technol. 4 (2017), 2851–2854.

[17] D. Poola, M.A. Salehi, K. Ramamohanarao, R. Buyya, A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments, Software Architecture for Big Data and the Cloud, Elsevier, Amsterdam, The Netherlands, 2017, pp. 285–320.

[18] M.G. Noll, Benchmarking and stress testing an Hadoop cluster with TeraSort, TestDFSIO & Co., 2011, Available from: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/.