

# EDAS: Efficient Data Access Scheme of Data Replication for Hadoop Distributed File System (HDFS)

Hnin Htet Htet Aung<sup>1</sup>, and Nyein Nyein Oo<sup>2</sup>

<sup>#</sup>Department of Information Technology Engineering, Yangon Technological University, Myanmar

<sup>1</sup>hninhtet8184@gmail.com and <sup>2</sup>nno2005@gmail.com

**Abstract:** Cloud computing is composed of a large number of distributed computation and storage resources to facilitate the management of distributed and sharing data resources efficiently. It is a great challenge to ensure efficient access of data replication to such huge and widely distributed data in cloud computing. To address this need, we proposed an Efficient Data Access Scheme (EDAS) of data replication for Hadoop Distributed File System (HDFS) to adaptively select the replica of data file form among service nodes. HDFS is an open source cloud based storage platform and deigned to be deployed in low-cost commodity hardware. In HDFS, data are distributed and replicated in cluster of commodity nodes. EDAS supports the access nodes decision of replica data for the users to get quick access form the adaptive services nodes according to the load of nodes. Aiming to provide the high performance of replication access and achieve load balance of service nodes, the proposed EDAS Algorithm implements based on historical data access record form the metadata of HDFS and anti-blocking probability selection method.

**Keywords:** cloud computing, Hadoop Distributed File System, blocking probability, replication access, load balance

## 1. Introduction

Cloud computing is a large-scale parallel and distributed computing system. It is an emerging paradigm that provides computing resources as a service over a network. Communication resources often become a bottleneck in service provisioning for many cloud applications. Therefore, data replication, a technique of creating multiple copies of an object such as data, file, database and so on, is seen as a promising solution [1]. It is commonly used to improve data availability, throughput and response time for user while it plays an important role for storage and access system. It allows minimizing network delays and bandwidth usage. For cloud environment in which data availability is a critical factor to improve system performance, replication is the key to improve the performance of cloud computing so that services can be provided to users as an agreement of SLAs (Service Level Agreements) [2].

Generally data replication methods address two common problems of replication which are replica allocation problem and replica placement problem. An approach which is used to determine how many replicas to allocate for each file and where to place them is called replica allocation problem. The management of replicas is critical for storage efficiency and data availability. Placing data as close as possible to computation is a common practice of data-intensive systems. In addition, different data blocks which are concurrently accessed by users are placed different nodes which is called replica placement problem.

The replication methods and strategies largely affect the performance of a distributed storage system. In cloud computing environment, data replication has been widely used as a mean of increasing the data availability of storage systems such as Redundant Arrays of Inexpensive Disks (RAID), Google file system [3], Ceph file system [4] and HDFS [5]. Moreover, the techniques of data replication are used to decrease the costs for cloud-related companies. Besides, the data replication can be used to reduce load balance of nodes and improve system consistency. It is of interest to know if an effective access scheme of data replication can achieve high performances of storage and load balance in cloud computing.

The rest of this paper is organized as follows. On Section 2 describes related work on data replication and data access of cloud computing system. We present background theory and proposed system architecture in Section 3 and section 4. Proposed EDAS and Access Frequency Algorithms are presented in Section 5. Calculation processes are given in Section 6 and conclusions and future work is described in Section 7.

## 2. Related work

There have been a number of research efforts in recent years to improve access frequency in large scale cloud storage system. Ruay-Shiung Chang et al., presented a dynamic replicating strategy, called Latest Access Largest Weight (LALW), to solve the problem of unbalance between work load of nodes and to improve data access in distributed system. First, through the concept of half- life, they find more popular files according to the time intervals. Next, they calculate the required replications of the files and allocate them. This method not only is suitable for the grid network, but also enhances the overall performance of system by allocating replications in more popular nodes. However, the method causes two problems: First, it creates the block on the node and is not suitable for cloud computing, which requires an environment of a large number of services. Second, it also causes the longer response time of system [2].

In adaptive data replication for efficient cluster scheduling (DARE), the authors proposed a dynamic data replication scheme based on access patterns of data blocks during runtime to improve data locality. Note that the default Hadoop distribution provides the fixed data replication in the phase of data storing. DARE allows to increase the data replication factor automatically by replicating the data to the fetched node. However, removing the replicated data is preformed when only the available data storage is insufficient. The data replication and placement algorithm adapts to the change in workload. Thus, it has a limit to provide the optimized replication factor with data access pattern [6].

Wei, Q., et al., presented a cost-effective dynamic replication management scheme referred to as CDRM. This model is used to determine how much minimal replica should be maintained to satisfy availability requirement. Replica placement is based on capacity and blocking probability of data nodes. In CDRM, blocking probability is used as a criterion to place replicas among data nodes to reduce access skew, so as to improve load balance and parallelism. But, name node used B+ tree Algorithm to sort data nodes in descending order using their blocking probability to place replica. This method isn't good for very large file whose size is Terabyte [7] and reduces the performance of the system.

Although some dynamic data replication mechanisms are suitable for cloud computing environment, they cause unbalances between nodes and degrades the performance of the system. In order to solve these problems, this system provides efficient access scheme to access replicas according to the workload of nodes. Thus, the motivation of this system is to improve access efficiency and availability for large scale data to support distributed computing environment, and provide the high performance of replication access and achieve load balance of service nodes. Because of the EDAS, the load of the system can be balanced and the performance of the system can be enhanced for any distributed file system.

## 3. Background Theory

With the introduction of cloud storage and cloud servers, it has become easier than ever to backup all important computer files online. To build such a cloud storage system, an increasing number of companies and academic institutions have started to rely on the Hadoop Distributed File System (HDFS). HDFS provides reliable storage and high throughput access to application data. It has been widely used and become a common storage appliance for cloud computing [8].

HDFS, an open-source framework, becomes a representative cloud storage platform. It is a distributed, high fault-tolerant file system designed for storing very large scale data with streaming data access patterns, running on clusters consisting of low-cost commodity hardware. It is divided into two kinds of nodes operating with a master-slave pattern: a NameNode and many DataNodes. The NameNode maintains the file system namespace and the metadata of all directories, files and blocks including block size, file length, replication factor, modification time, ownership, permission information, etc.

DataNodes provide block storage, server I/O requests from cloud users and perform block operations upon instructions from NameNode. In Hadoop cloud storage infrastructure, data are divided into fixed-sized blocks (64 MB each) which are stored as independent units. Each block is replicated to the DataNodes for fault

tolerance. Specifically, users can define replication factor when creating a file. By default, every file has a same replication factor which equals three. Data are spread in a number of DataNodes in Hadoop cloud storage system and often need to communicate with DataNodes for store and access the data operation.

An HDFS client wanting to read a file first contacts the NameNode for the locations of data blocks comprising the file and then reads block contents from the DataNode closest to the client. When writing data, the client requests the NameNode to nominate a suite of three DataNodes to host the block replicas. The client then writes data to the DataNodes in a pipeline fashion. The current design has a single NameNode for each cluster. The cluster can have thousands of DataNodes and tens of thousands of HDFS clients per cluster, as each DataNode may execute multiple application tasks concurrently [3][4] [9].

#### 4. Proposed System Architecture

The proposed scheme includes three-stage processes: we calculate Access Frequency (AF) with the proposed algorithm in the first stage and choose the node with lower Blocking Probability (BP) value as a second stage and then give the result node decision for the user from the NameNode after comparing AF of each elected node in the third stage. Fig. 1 describes the proposed system architecture.

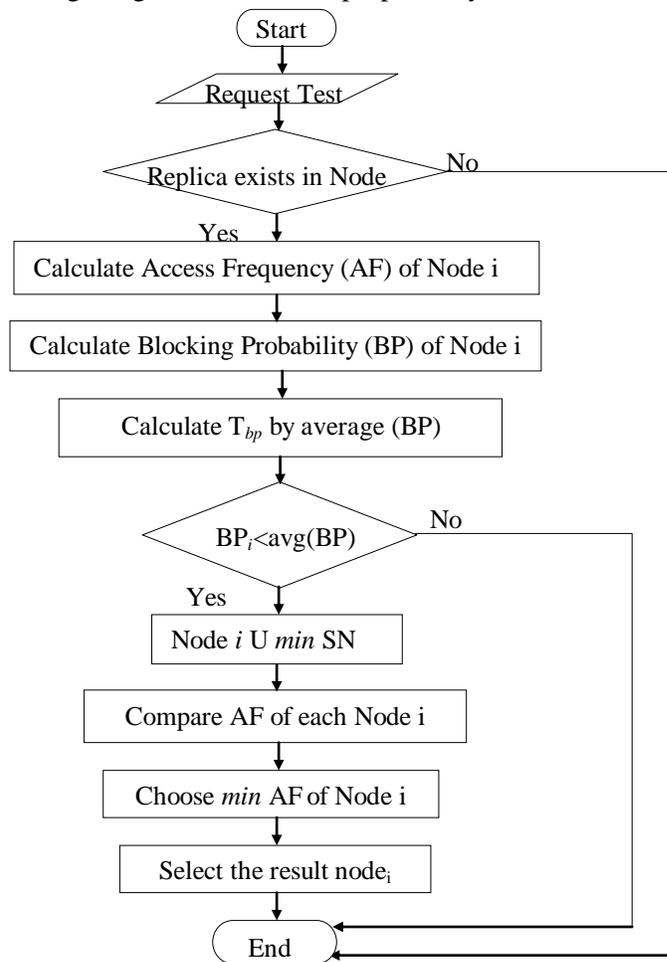


Fig. 1: Flowchart of Proposed EDAS

##### 4.1. Proposed Access Frequency Algorithm

Access Frequency is calculated by using audit log file from the metadata of HDFS. The HDFS user audit log file is divided into a number of small files based on logging time, which will later be called as time window (tw). In each tw, the access frequency is counted and stored for individual files. Then access frequency counts of each time window are aggregated by file name. Proposed AF Algorithm is used to calculate access frequency values for each accessed file. An example for aggregation of access frequency (Access Time) records is shown in Fig. 2. Fig. 3 shows Proposed Access Frequency Algorithm.

Time window (tw <sub>1</sub> )			Time window (tw <sub>2</sub> )			Aggregate Access Time			Time window (tw <sub>3</sub> )		
File ID	Node ID	Access Time	File ID	Node ID	Access Time	File ID	Node ID	Access Time	File ID	Node ID	Access Time
A	DN <sub>1</sub>	1	A	DN <sub>1</sub>	2	A	DN <sub>1</sub>	4	A	DN <sub>1</sub>	1
B	DN <sub>1</sub>	2	B	DN <sub>1</sub>	1	B	DN <sub>1</sub>	6	B	DN <sub>1</sub>	3
C	DN <sub>3</sub>	4	C	DN <sub>3</sub>	1	C	DN <sub>3</sub>	7	C	DN <sub>3</sub>	2
A	DN <sub>2</sub>	1	A	DN <sub>2</sub>	2	A	DN <sub>2</sub>	5	A	DN <sub>2</sub>	2
B	DN <sub>4</sub>	3	B	DN <sub>4</sub>	3	B	DN <sub>4</sub>	9	B	DN <sub>4</sub>	3
C	DN <sub>4</sub>	1	C	DN <sub>4</sub>	1	C	DN <sub>4</sub>	3	C	DN <sub>4</sub>	1
A	DN <sub>3</sub>	2	A	DN <sub>3</sub>	3	A	DN <sub>3</sub>	8	A	DN <sub>3</sub>	3
B	DN <sub>5</sub>	2	B	DN <sub>5</sub>	2	B	DN <sub>5</sub>	7	B	DN <sub>5</sub>	3
C	DN <sub>5</sub>	1	C	DN <sub>5</sub>	1	C	DN <sub>5</sub>	4	C	DN <sub>5</sub>	2

Fig. 2: An example for aggregation of access frequency (Access Time) records

TABLE I: Notations Used in Access Frequency Algorithm

Notation	Description
AF <sub>x</sub>	The access frequency of requested file x
tw <sub>j</sub>	Time window or logging time
DN <sub>i</sub>	Data Node i
inLog	The input log file
j	Number of time window {tw <sub>1</sub> , tw <sub>2</sub> , ..., tw <sub>n</sub> }
i	Number of data node {DN <sub>1</sub> , DN <sub>2</sub> , ..., DN <sub>3</sub> }
x	Access file name (variable)

**Proposed Access Frequency Algorithm**

✓ input : inlog, file x  
 ✓ output : AF<sub>x</sub>

Begin

1. Define tw size
2. Send request file x
3. while (tw < tw size)
  - {
  - 1. Read inlog
  - 2. if access file x exit in inlog
    - { Calculate AF of each file for DN<sub>i</sub> by using

$$AF_x(DN_i) = \sum_{j=1}^n AF_{tw_j(x)}(DN_i)$$

- 
- 
- 
4. return AF<sub>x</sub>

End

Fig. 3: Proposed Access Frequency Algorithm

## 4.2. Proposed EDAS Algorithm

In the proposed system, when the user requests a task, the NameNode searches the DataNodes which have the replica of the requested task exist. Then, we calculate the access frequency value from the metadata of HDFS and the blocking probability value of each node based on the total arrival rate and delay time. After calculating, we select nodes which have blocking probability lower than average blocking probability.

After that, the node with the smallest access frequency value is picked as the appropriate node for the user to access the file. This algorithm mainly supports for the user to get quick access from the adaptive service nodes according to the workload of nodes to improve access efficiency and performance of the system. Fig. 2 shows the flowchart of efficient data access scheme of data replication for HDFS. TABLE II describes the notation and used in EDAS. Fig. 4 shows the proposed EDAS Algorithm.

TABLE II: Notations Used in EDAS Algorithm

Notation	Description
$file_x$	The user requested file
$AF_i$	The Access Frequency/ Access Time (AF) of node i
$BP_i$	The Blocking Probability (BP) value of node i
avg (BP)	The average BP of all nodes
minSN	The set of nodes with $BP_i$ lower than the avg (BP)
min(AF)	The lowest AF value among all nodes
$AN_i$	Resulted Access Node i

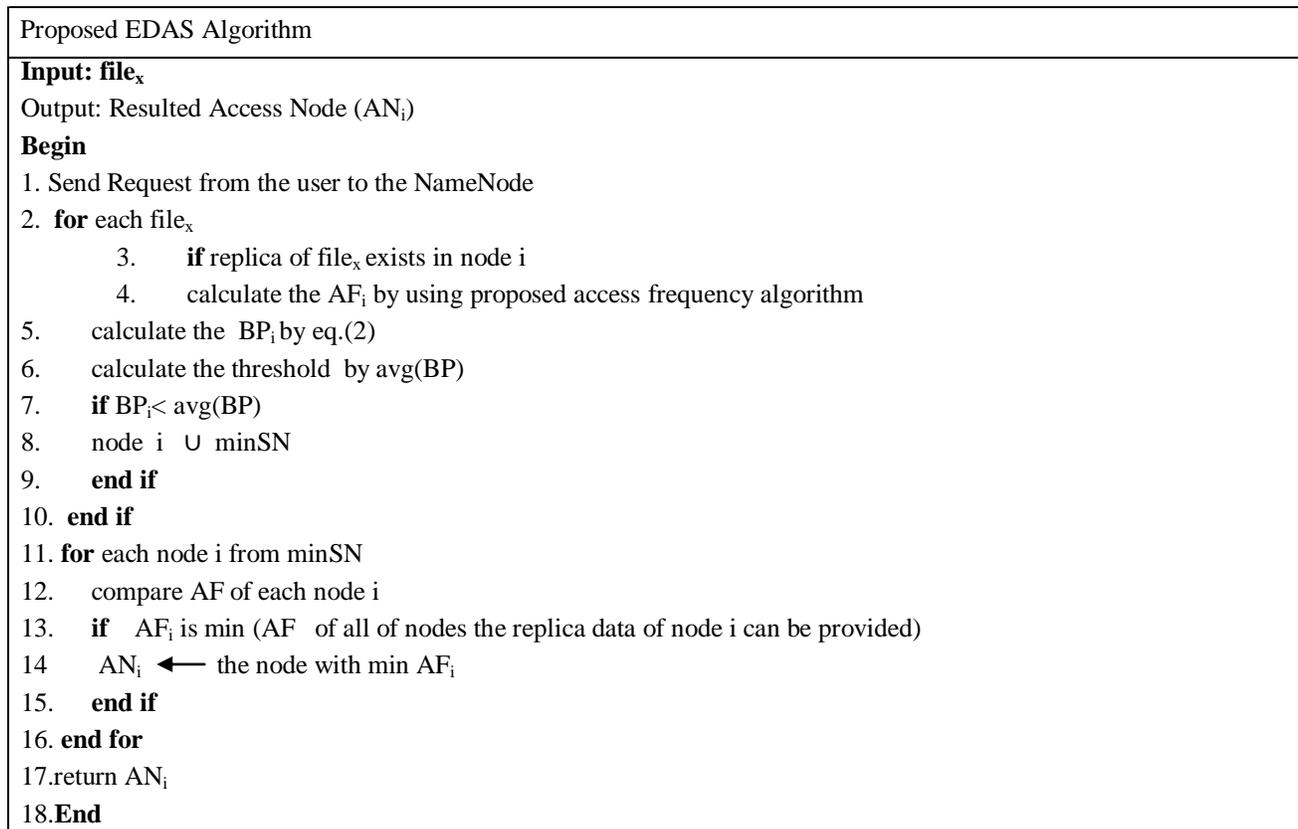


Fig. 4: Proposed EDAS Algorithm

## 4.3 Anti-Blocking Probability Selection (ABPS)

The ABPS is based on arrival rate ( $\lambda$ ) that is calculated by Eq. (1) and the delay time ( $\tau$ ) to select the service node with lower blocking probabilities. The high blocking probabilities of nodes cause the data losses and reduce the performance.

$$\text{Arrival Rate } (\lambda_i): \lambda_i = (\rho_j / \gamma_j) \lambda \quad (1)$$

where,  $\gamma_j$  : the number of replicas,

$\rho_j$ : the popularity of access file

$i$ : time for incoming task

$j$ : number of nodes belong to access file

$\lambda$ : the number of requests which is actually arrive the cluster, and it is called arrival rate

The first stage designates the node location of replicas by calculating Blocking Probability ( $BP_i$ ) by using Eq. (2) and access data in replica strange nodes with lower  $T_{bp}$  to avoid the block and time delay. In order to reduce the request losses, the access times replicas are adjusted to share the loads in those sub-nodes which have BP higher than BP threshold value ( $T_{bp}$ ). ( $T_{bp}$ ) is calculated by average of BP.

$$BP = \frac{(\lambda_i \tau_i)^{c_i}}{c_i!} \left[ \sum_{k=0}^{c_i} \frac{(\lambda_i \tau_i)^k}{k!} \right]^{-1} \quad (2)$$

where,  $c_i$  means multiple sessions that is divided by accessed memory and  $\tau_i$  is delay time of each sub-node. Blocking means when  $c_i$  is fully occupied and puts the new incoming files waiting. After selecting replica-nodes with lower BP, this stage balances the loading rate of nodes. EDAS is used to provide high performance of replication access rely on BP and AF. In Fig.4, min SN means the set of nodes with BP lower than avg (BP).

## 5. Calculation Processes of Proposed EDAS Algorithm

In this section, we present how to select the node which is suitable for accessing the desired file by using the proposed EDAS Algorithm. The EDAS can select replica-nodes with lower BP to reduce the delay and response time through the weighted value of AF. The needs of users can be distributed evenly on each node and resources of nodes can be fully utilized. In the following instance, the Name node is assumed to provide different replicas of files through five Data nodes: N1, N2, N3, N4, N5; the files A, B, C are accessed at the time point respectively.

Assume the customer requests to access file A, we calculate the AF value of requested file A by using proposed AF Algorithm. In this example, we use AF value from Fig.2. Now, we can define each parameter of “Eq. (1)” in more details: $\lambda$ : we assumed that the total number of requests is 300, and only 290 requests delivered according to the concept of “possession”. Other request may be failed due to pack losses or long waiting time during the delivery from the customer. Therefore,  $\lambda$  is assumed to be 0.967.

We have to calculate from all nodes file A. Then we calculate the arrival rate of selected node ( $\lambda_i$ ) and BP values of elected nodes based on the total arrival rate ( $\lambda=0.967$ ); and obtain the individual arrival rates of each nodes based on the number of replicas within the nodes. Now we use the “Eq. (2)” of EDAS, to calculate the BP. Table III shows the results of arrival rate ( $\lambda_i$ ) and blocking probabilities  $BP_i$ .

TABLE III: Arrival Rate ( $\lambda_i$ ) and Blocking Probabilities (BP) at Each Node

NodeID	AF <sub>i</sub>	$\lambda_i$	BP <sub>i</sub>
DN <sub>1</sub>	4	0.68	0.22
DN <sub>2</sub>	5	0.85	0.29
DN <sub>3</sub>	8	1.37	0.46

After calculation, we select nodes based on the average BP values:  $(0.22+0.29+0.46)/3=0.323$

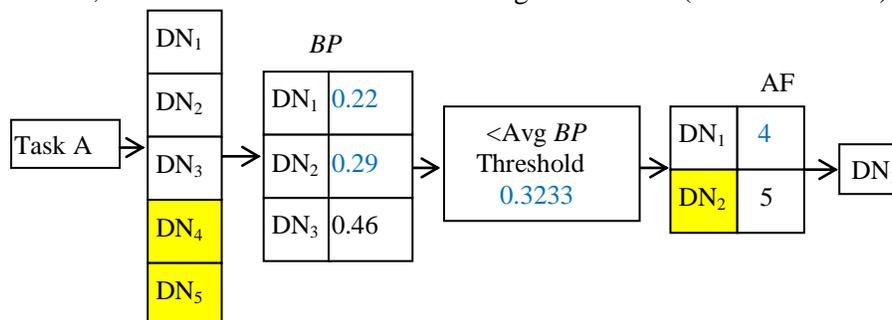


Fig. 5: Example of EDAS

By setting up the average BP value as the threshold value ( $T_{bp}$ ), the node which has BP smaller than the threshold value can be elected. For the above example, the BP values of node 1 and 2 are lower than the threshold value. As the first phase, we choose node 1 and 2. We then compare those nodes based on their total AF and pick the node with the lowest AF as the service node for the current period. Now, node 1 and 2 are compared. Then node 1 is selected as the service node since it has the smallest AF. Fig.5 shows example of how to select the nodes with BP and AF values.

In this example, we choose the node with minimum AF and lowest BP values to avoid data lost and longer response time. This system is essentially responsible for deciding the most suitable node for the user to get quick access depending on the workload of nodes to provide high performance of replication access and load balance in cloud computing environments. We propose EDAS for the task with first access and without conflict in hadoop distributed file system.

## 6. Conclusion and Future work

Proposed scheme dispatches the task from the adapted services nodes based on lower blocking probability and lower access time. EDAS gives better access, load balance and system performance than the original access operation of HDFS. This system supports not only static but also dynamic replication strategies for any distributed system. The selected nodes have to process request endlessly thus cause the unbalancing loading, traffic jam and slower response in the traditional access system. To overcome these conditions, the proposed system provides the high performance of replication access and achieves the overall load balance of service nodes according to the workload of services nodes. As the future work, we will implement the complete system and evaluate the performance of the replica access with relevant graphs with time function. We will compare the access time of original HDFS operation and proposed system to ensure efficient access of data replication to such huge and widely distributed data in cloud computing.

## 7. References

- [1] Mohamed-K HUSSEIN, Mohamed-H MOUSA, "LDRC: A Light-weight Data Replication for Cloud Data Centers Environment." *International Journal of Engineering and Innovative Technology (IJEIT)*, Vol 1, Issue 6, June 2012.
- [2] R.-S. Chang and H.-P. Chang, "A dynamic data replication strategy using access-weights in data grids." *J. Supercomput.*, vol. 45, no. 3, pp. 277–295, Jan. 2008.  
<http://dx.doi.org/10.1007/s11227-008-0172-6>
- [3] The Apache Software Foundation. Hadoop. <http://hadoop.apache.org/core/>, 2009.
- [4] Amazon-S3. Amazon simple storage service (amazon s3). <http://www.amazon.com/s>, 2009.
- [5] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur. Pvfs: "A parallel file system for linux clusters." *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, USENIX.
- [6] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling," *2011 IEEE Int. Conf. Clust. Comput.*, pp. 159–168, Sep. 2011.
- [7] Wei, Q.S., Veeravalli, B., Gong, B., Zeng, L.F. & Feng, D. 2010. "CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster." *IEEE International Conference on Cluster Computing (CLUSTER)*, Crete: Heraklion, 188-196.  
<http://dx.doi.org/10.1109/CLUSTER.2010.24>
- [8] Tin Tin Yee and Thin Thu Naing, "An efficient Data Prefetching Technique for PC-Cluster Based Cloud Storage System." *12<sup>th</sup> International Conference on Computer Applications* 2014, 2014.
- [9] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System", *Sunnyvale, California USA {Shv, Hairong, SRadia, Chansler}@Yahoo-Inc.com. IEEE*, 2010.  
<http://dx.doi.org/10.1109/MSST.2010.5496972>