Dynamic Replication Management Scheme for Cloud Storage

May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye
University of Computer Studies, Yangon
mayphyothu.mpt1@gmail.com, khinemoenwe@ucsy.edu.mm, kyarnyoaye@gmail.com

Abstract

Nowadays, replication technique is widely used in data center storage systems to prevent data loss. Data popularity is a key factor in data replication as popular files are accessed most frequently and then they become unstable and unpredictable. Moreover, replicas placement is one of key issues that affect the performance of the system such as load balancing, data locality etc. Data locality is a fundamental problem to data-parallel applications that often happens (i.e., a data block should be copied to the processing node when a processing node does not possess the data block in its local storage), and this problem leads to the decrease in performance. To address these challenges, this paper proposes a dynamic replication management scheme based on data popularity and data locality; it includes replica allocation and replica placement algorithms. Data locality, disk bandwidth, CPU processing speed and storage utilization are considered in the proposed data placement algorithm in order to achieve better data locality and load balancing effectively. Our proposed scheme will be effective for large-scale cloud storage.

Keywords- Replication, Data Popularity, Data locality, Storage utilization, Disk Bandwidth

1. Introduction

Cloud storage is a technology that allows us to save files in storage and then access those files via Cloud. The cloud storage system convergences data storage among multiple servers into a single storage pool and provides users with immediate access to a broad range of resources and applications hosted in the infrastructure of another organization via a web service interface [6].

Cloud storage systems may consist of a cluster of storage nodes or even geographically distributed data centers. At present, the existing Cloud storage products are Google (Google File System GFS), Amazon (Simple Storage Service S3), IBM (Blue Cloud), Yahoo (Hadoop Distributed File System HDFS) etc. HDFS provides reliable storage and high throughput access to application data. In HDFS, data is split in a fixed size (e.g., 32MB, 64MB, and 128MB) and the split data blocks (chunks) are distributed and stored in multiple data nodes with replication.

In HDFS, to provide data locality, Hadoop tries to automatically collocate the data with the computing node. Hadoop schedules Map tasks to set the data on same node and the same rack. Data locality is a principal factor of Hadoop's performance. The data locality problem occurs when the assigned node should load the data block from a different node storing the data block. Data locality means the degree of distance between data and the processing node for the data.

There are two ways in order to improve data locality:

1. The replica allocation problem occurs when popular data are assigned a larger number of replicas to improve data locality of concurrent accesses.
2. The replica placement problem occurs when different data blocks accessed concurrently are placed on different nodes to reduce contention on a particular node.

There are three types of data locality in Hadoop: node locality, rack locality and rack-off locality. Uniform data replication is used in current implementations of MapReduce systems (e.g., Hadoop). The concept of popularity of files is introduced to replication strategies for selecting a popular file in reality. File popularity represents whether a file has been hot in recent time intervals, which is computed by file access rate.

In this paper, therefore, data popularity based replication method is proposed to overcome the problems of static replication in HDFS and to support better efficiency in cloud storage. Firstly, the rate of change of file popularity is calculated by analyzing the access histories with first order differential equation. Secondly, the replication degree for each file is calculated according to the rate of change of file popularity. Finally, the replicas will be placed based on proposed data placement algorithm.

The rest of this paper is organized as follows. Section 2 describes related works and background theory is presented in section 3. Section 4 presents proposed system architecture and finally, section 5 describes the conclusion and future work.

2. Related Works

In cloud storage environment, data can be stored with some geographical or logical distance and this data is accessible to cloud based applications. Data is replicated

ISBN 978-99971-0-381-9

and stored in multiple data nodes to provide high availability and load balancing. There were several previous researches of data replication in HDFS. A cost effective replication management scheme for cloud storage cluster was proposed by Qingsong Wei [2]. That paper aimed to improve file availability by centrally determining the ideal number of replicas for a file, and an adequate placement strategy based on the blocking probability. However, this method wasn't good for very large file that was file size was Terabyte and the effects of increasing locality were not studied.

One approach, Latest Access Largest Weight (LALW) algorithm [8], that was proposed by R.S. Chang and H.P.Chang for data grids. LALW found out the most popular file in the end of each time interval and calculated a suitable number of copies for that popular file and decides which grid sites were suitable to locate the replicas.

A. Hunger and J. Myint compared two data popularity-based replication algorithms: PopStore and Latest Access Largest Weight (LALW) [1]. In that paper, both algorithms found more popular files according to the time intervals through the concept of Half-life. However, this paper did not consider for load balance in replica placement.

Recently, a few studies attempted to improve data locality with data replication in Hadoop. Scarlett [5] adopted a proactive replication scheme that periodically replicates files based on predicted data popularity. It focused on data that receives at least three concurrent accesses. However, it did not consider node popularity caused by co-location of moderately popular data.

In DARE[3], the authors proposed a dynamic data replication scheme based on access patterns of data blocks during runtime to improve data locality. DARE adopted a reactive approach that probabilistically retained remotely retrieved data and evicted aged replicas. DARE allowed to increase the data replication factor automatically by replicating the data to the fetched node. However, removing the replicated data was performed when only the available data storage was insufficient. Thus, it had a limit to provide the optimized replication factor with data access patterns.

In [9], the authors proposed a delay scheduling method that focused on the conflict between data locality and fairness among jobs. Although the delay scheduling method was designed to improve data locality, it let the jobs wait for a small amount of time, resulting in violating the fairness for jobs. Moreover, delay scheduling made assumptions that might not hold universally: (a) task durations were short and bimodal, and (b) a fixed waiting time parameter worked for all loads and skewness of traffic. These assumptions made it difficult for delay scheduling to adapt to changes in workload, network conditions, or node popularity.

In [4], the authors proposed an efficient data replication scheme based on access count prediction in a Hadoop framework. This data replication scheme determined the replication factor with the predicted data access count, whether it generated a new replica or it used the loaded data as cache selectively. Although this scheme was designed to improve data locality, it considered file level replication did not consider block level replication.

3. Background Theory

In large-scale distributed system, replication is a general technology that can improve the efficiency of data access and the fault-tolerance. Data locality is a principal factor of Hadoop's performance. In Hadoop scheduling policy, the data locality problem occurs; that is, the assigned node should load the data block from a different node storing the data block. The proposed dynamic replication management scheme considers the data popularity and data locality. This section describes architecture of Hadoop cluster and data locality.

3.1 Architecture of Hadoop Cluster

Hadoop is an open source software framework that supports data intensive distributed applications. The architecture of a Hadoop cluster can be divided into two layers: MapReduce and HDFS (Hadoop Distributed File System). The MapReduce layer maintains MapReduce jobs and their tasks, and the HDFS layer is responsible for storing and managing data blocks and their metadata. HDFS stores three replicas of each block like Google File System (GFS) [7].

A job tracker in the master node splits a MapReduce job into several tasks and the split tasks are scheduled to task trackers by the job tracker. For the purpose of monitoring the state of task trackers, the job tracker aggregates the heartbeat messages from the task trackers. When storing input data into the HDFS, the data are split in fixed sized data blocks with replication (the default replication factor is 3) and the split data blocks (chunks) are stored in slave nodes. A task tracker of a slave node is in charge of scheduling tasks in the node. A task tracker requests a task from a job tracker by sending a heartbeat message when it has an empty task slot.

When storing input data from a client, the data are divided into chunks and the chunks are stored to nodes. The job tracker deals with a MapReduce job request from a client. Upon reception of a job request, the job tracker divides a job into tasks, and then, the tasks are assigned to task trackers. At this stage, it schedules tasks by considering data locality. Next, each task tracker assigns a task to a node, and then, the node performs the task by loading the data block from HDFS when needed.

Users submit jobs consisting of a map function and a reduce function. Hadoop breaks each job into tasks. First, input data are divided into fixed size units processed independently and in parallel by map tasks, which are executed distributively across the nodes in the cluster. There is one map task per input block. After the map tasks are executed, their output is shuffled, sorted and then processed in parallel by one or more reduce tasks.

3.2 Data Locality

Data in Hadoop is stored in HDFS. This data is divided into blocks and stored across the data nodes in a Hadoop cluster. When a MapReduce job is executed against the dataset, the individual Mappers will process the blocks (input splits). When data is not available for Mapper in the same node, then data has to copied over the network from the data node that has data to the data node that is executing the Mapper task. This is known as a data locality.

Data locality related with the distance between data and the processing node. So, if the closer distance between data and node, it has the better data locality. There are three types of data locality in Hadoop:

(1) Node locality: when data for processing are stored in the local storage,
(2) Rack locality: when data for processing are not stored in the local storage, but another node within the same rack,
(3) Rack-off locality: when data for processing are not stored in the local storage and nodes within the same rack, but another node in a different rack.

Figure 1 shows three types of data locality in Hadoop: node locality, rack locality, and rack-off locality.

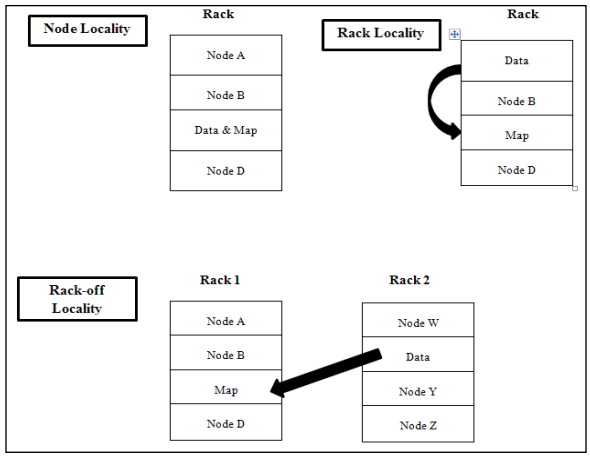

Figure 1. Types of data locality

Among these types of data locality, the most preferred scenario is node locality and the least preferred scenario is rack-off locality. The data locality problem is a situation where a task is scheduled with rack or rack-off locality. Moreover, the overhead of rack-off locality is greater than that of rack locality. To prevent the data locality problem, we propose a dynamic data replication scheme using prediction by the access count of data files and a data placement algorithm reducing case of rack and rack-off locality.

4. Proposed System Architecture

The basic idea of replication is based on the different replication degree per data file. Maintaining the static number of replicas in the system results highly storage cost for unpopular data and inefficient for most accessed data. Moreover, maintaining too much replication degree than the current access count for a data file does not always guarantee the better data locality for all data blocks.

The goal of proposed system is to design an adaptive replication scheme that seeks to increase data locality by replicating "popular" data while keeping a minimum number of replicas for unpopular data. Because the nature of data access pattern is random, a method that predicts the rate of change of file popularity for the next time slot is required. The proposed system flow diagram is shown in figure 2.

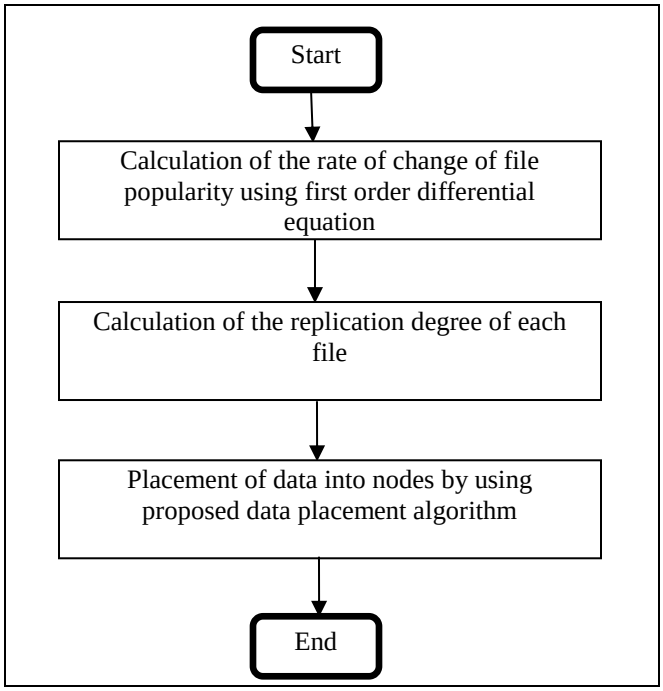

Figure 2. Proposed system flow diagram

The proposed scheme includes three-step processes: , the rate of change of file popularity will be calculated using first order differential equation in the first step and

the number of replicas of each file will be calculated in the second step and then the replicas will be placed into nodes based on proposed data placement algorithm in the third step.

4.1. Proposed Popularity Growth Rate Algorithm

In this step, the rate of change of file popularity will be calculated using first order differential equation. LALW and Pop-Store algorithms applied half-life strategy which means the weight of the records in an interval decays to half of its previous weight.

The idea of popularity is the assumption that the rate at which a popularity of an item grows at a certain time is proportional to the total popularity of the item at that time. In mathematical terms, if *P(t)* denotes the total population at time t, then this assumption can be expressed as

$$\frac{dp}{dt} = kP(t) \qquad (1)$$

where *P(t)* denotes population at time t and k is the growth constant or the decay constant, as appropriate. If k > 0, we have growth, and if k <0, we have decay. It is a linear differential equation which solve into

$$P(t) = P_0 e^{kt} \qquad (2)$$

Then,

$$k = \frac{\ln(\frac{P(t)}{P_0})}{t} \qquad (3)$$

Where P_0 is the initial population, i.e. p (0) = P_0, and k is called the growth or the decay constant. In this step, the rate of change of file popularity will be calculated by using Yahoo Hadoop audit log file [10] as data source. Users can enable audit logging from the NameNode. Audit events are emitted as a set of key/value pairs for the following keys as shown in table 1.

Table 1. Key/Value Pairs of User Audit Log

Key	Value
ugi	<user>, <group>[,<group>]*
ip	<client ip address>
cmd	(open|create|delete|rename|mkdirs|listStatus| setReplication|setOwner|setPermission)
src	<path>
dst	(<path>|"null")
perm	(<user>:<group>:<perm mask>|"null")

The Yahoo HDFS User Audit log format is shown in figure 3.

2016-12-10 11:11:59,693 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem. audit: **ugi**=hduser **ip**=/134.91.100.59 **cmd**=delete **src**=/app/hadoop/tmp/test.txt **dst**=null **perm**=null

Figure 3. HDFS user audit log format

To get the frequency count of each file, user audit log is split into small files based on timeslot duration and number of records. Then the required fields are extracted such as Date, Time, IP and src. After that, the user access frequency is counted from src source link from figure 4. In each time slot, the access frequency is counted and stored for individual files. Then the rate of change of file popularity of each file is calculated on each time slot according to table 2 and algorithm 1.

Table 2. Notations Used in Popularity Growth Rate Algorithm

Notation	Description
$P(t_f)$	Popularity values of file f
$AF(t_f)$	Total access frequency of file f at each time slot
inLog	The input log file
k	The rate of change of file popularity

Algorithm 1: Popularity Growth Rate Algorithm

Input: inLog
Output: *k*

1. Read inLog
2. Calculate access frequency of each file by using
   $P(t_f) = AF(t_f)$, $\forall$ f $\in$ F
3. Calculate the rate of change of file popularity k of each file by substituting $P(t) = P(t_f)$ in (3)
4. return k.

Figure 4. Popularity growth rate algorithm

In order to verify our proposed popularity growth rate algorithm, we suppose three files (x_1, x_2 and x_3) in three time slots. Each time slot duration is set as 10 seconds, therefore, (t1= t2 = t3= 10 seconds). Let P_0 = 1, P (t) = $AF(t_f)$ and calculate k by using equation (3). Suppose access frequencies of file x1, x2 and x3 in time slot 1 are 40, 1100 and 200. In time slot 1, the rate of change of file popularity k in file x1, x2 and x3 is 0.3688, 0.7003 and 0.5298. Also, in time slot 2, access frequencies of file x1, x2 and x3 are 400, 100 and 900. Therefore, the rate of change of file popularity k in file x1, x2 and x3 for time slot 2 is 0.5991, 0.4605 and 0.6802. Also, in time slot 3, access frequencies of file x1, x2 and x3 are 2200, 1200 and 20. Therefore, the rate of change of file popularity k

in file x1, x2 and x3 for time slot 3 is 0.7696, 0.7090 and 0.2996.

According to the rate of change of file popularity, replica degree for each file is considered as follows. If the rate of change of file popularity is greater than 0.0, then existing replica degree is increased by one. If the rate of change of file popularity is less than 0.0, then existing replica degree is decreased by one. If the rate of change of file popularity is equal to 0.0 then existing replica degree is remained unchanged. Otherwise, if the accessed file is new and there is no access record history, the replica degree for this file will be assigned 3 as like HDFS default replica number.

4.2. Proposed Data Placement Algorithm

After determining the number of replicas, we will consider how to place these replicas efficiently in order to improve data locality and load balancing. In this step, let me assume that the jobs will have to access this replica in the next time slot. The incoming job is broken into tasks and each map task is assigned into nodes within the cluster. There is one map task per input block.

In this system, the input data file is divided into 64 MB blocks and place them into blocks within the cluster. For instance, if the replica for this file is 3 and this file has 4 blocks, then the total replica block number of this file is 12. Let the maximum number of replicas be the number of nodes in the cluster and the minimum number of replicas be 1.

Suppose that at the assigned node, there is no replica block for the incoming map task. In this case, this system considers for improvement of node locality. In this case, the remote data retrieval is performed by loading the replica data block into this node. While loading this data block, if the load factor of this node is less than the predefined threshold, this replica data block is loaded into this node. Otherwise, the replacement is performed by replacing this replica data block with existing block into this node.

The proposed data replacement algorithm is based on Least Recently Used (LRU). It will be more reliable than the LRU and will have the more efficient results than the LRU algorithm because it considers access frequency for replacement. Firstly, this proposed algorithm considers the block with minimum access frequency for replacement. Secondly, if one or more blocks with minimum access frequency, it considers least recently accessed block (outgoing block) for replacement according to LRU mechanism. The proposed enhanced LRU replacement algorithm is shown in figure 5.

Algorithm 2: Enhanced LRU Algorithm

Step 1. When loading the replica data block into the assigned node, it will calculate total number of access frequencies (TAF) for all blocks in that node.

Step 2. If only one block with minimum TAF is found, that block will be selected to evict from that node.

Step 3. If one or more minimum TAF blocks are found, least recently accessed block (outgoing block) will be selected to evict from that node as LRU.

Figure 5. Enhanced LRU Algorithm

The existing Hadoop block placement strategy does not take into account DataNodes' utilization, which leads to in an imbalanced load. Since the DataNode selection for the block placement is random, the disk bandwidth of the allotted DataNode may be less than or greater than the available bandwidth. This policy assumes that all nodes in the cluster are homogeneous, and randomly place blocks without considering any nodes' resource characteristics, which decreases self-adaptability of the system. Therefore, this system considers the heterogeneous environment for nodes in the cluster. We need to consider the load factor such as storage utilization, disk bandwidth and CPU processing speed. During the process of placement, the storage utilization, disk bandwidth and CPU processing speed of DataNode are important factors to affect the load balancing in HDFS. Therefore, the capacity of DataNode stored should be proportional to its total disk capacity, in the condition of effective load balancing. We can carry out the storage utilization model as

$$U(D_i) = \frac{D_i\ (use)}{D_i\ (total)} \qquad (4)$$

Where,$U(D_i)$ is the storage utilization of the ith DataNode. $D_i(use)$ is the used disk capacity of the ith DataNode, and its unit is GB. $D_i(total)$ is the total disk capacity of the ith DataNode, it is a fixed value of each DataNode, and its unit is GB.

Then, we can carry out the disk bandwidth model as

$$BW(D_i) = \frac{T_b}{T_s} \qquad (5)$$

Where, $BW(D_i)$ is the disk bandwidth of the ith DataNode. T_b is the total number of bytes transferred, and T_s is the total time between the first request for service and the completion of the last transfer.

Then, the CPU processing speed is used as one of the important factors and each node has different CPU processing speed due to the heterogeneous environment.

Among these three factors, storage utilization is set as first priority, disk bandwidth as second priority and CPU processing speed as last priority. So, we put the coefficients of storage utilization, disk bandwidth and CPU processing speed are set as 0.5, 0.3 and 0.2. Therefore, we can carry out the load factor model as

$$LF(D_i) = 0.5U(D_i) + 0.3BW(D_i) + 0.2SP(D_i) \quad (6)$$

The predefined threshold T_i of the i^{th} cluster is assumed as the sum of maximum storage utilization, maximum disk bandwidth and maximum CPU processing speed in cluster is divided by the number of nodes in the cluster. Therefore, we can carry out the predefined threshold of cluster C_i as

$$T_i = \frac{Max_i\ (U) + Max_i\ (BW) + Max_i\ (SP)}{N} \quad (7)$$

Where, T_i is the predefined threshold of the i^{th} cluster and N is the number of nodes in the i^{th} cluster. If the load factor of this node is less than the predefined threshold, this replica data block is loaded into this node. Therefore, the storage utilization, disk utilization and CPU processing speed of DataNode are used in proposed data placement algorithm as shown in table 3 and algorithm 3.

Table 3. Notations Used in Data Placement Algorithm

Notation	Description
DN	DataNodes list
BW	Bandwidth
U	Storage utilization
RP	Replica List
MT	Map task list
SP	CPU processing speed
C	Cluster list
LF	Load factor list

5. Conclusion

In cloud storage environment, data can be stored with some geographical or logical distance and this data is accessible for cloud based applications. Data is replicated and stored in multiple data nodes to provide for data availability. In this paper, a dynamic replication management scheme is proposed for cloud storage. At each time intervals, the proposed system collects the data access history in cloud storage. According to access frequencies for all files that have been requested, the change of popularity rate can be calculated and replicated them to suitable DataNodes in order to achieve load balance and node locality of system. As a future work, many experimental evaluations have to be carried out in order to get the efficiency of proposed data placement algorithm. In addition, many experimental evaluations have to be performed in order to get better threshold value and load factor value. And as well, replica deallocation will be considered for overall system improvement.

Algorithm 3: Data Placement Algorithm

Input: DataNodes List DN= {DN_1, DN_2,.., DN_n }, Replica List RP ={ RP_1, RP_2, RP_3,…., RP_n }, Map Task List MT = {MT_1,MT_2,MT_3,…,MT_n}, Load Factor List LF = {LF_1,LF_2,LF_3,…., LF_n}, Predefined Threshold T_i, Cluster List C = {C_1, C_2, C_3,…., C_n}
Output: DataNodes List DN

for each incoming map task **MT do**
  for each DataNode **DN do**
    Check node locality of task **MT_i**
    if there is node locality **then** assign task **MT_i** to that DataNode **DN_i**
    else
      Perform remote data replica retrieval for task **MT_i**
      Calculate storage utilization **U** of this assigned DataNode **DN_i** using (4)
      Calculate disk bandwidth BW of this assigned DataNode **DN_i** using (5)
      Check CPU processing speed **SP** of this assigned DataNode **DN_i**
      Calculate load factor **LF_i** for this assigned DataNode **DN_i** using (6)
      Calculate predefined threshold **T_i** for the cluster **C_i**
      if LF_i > predefined threshold **T_i then**
      Perform replacement using algorithm 2
      Place replica **RP_i** for this task on that DataNode **DN_i**
      break
      else
       Place replica **RP_i** for this task on that DataNode **DN_i**
      break
    end if
  end if
 end for
end for

Figure 6. Data Placement Algorithm

6. References

[1] A. Hunger and J. Myint, "Comparative Analysis of Adaptive File Replication Algorithms for Cloud Data Storage", *2014 International Conference on Future Internet of Things and Cloud*, 2014.

[2] B. Gong, B. Veeravalli, D. Feng L. Zeng, and Q. Wei, "CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster", *2010 IEEE International Conference on Cluster Computing*, Sep. 2010, pp. 188–196.

[3] C.L. Abad, Yi Lu, R.H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling", *IEEE International Conference on Cluster Computing* (CLUSTER 2011), pp.159-168, 2011.

[4] D.Lee, J.Lee, and J.Chung , "Efficient Data Replication Scheme based on Hadoop Distributed File System",

International Journal of Software Engineering and Its Applications Vol. 9, No. 12 (2015), pp. 177-186,2015.

[5] G. Ananthanarayanan *et al.*, "Scarlett: Coping with skewed content popularity in mapreduce clusters," in *Proc. Conf. Comput. Syst. (EuroSys)*, 2011, pp. 287–300.

[6] H. Gobioff, S. Ghemawat, and S.-T. Leung, "The Google File System", *Proceedings of 19th ACM Symposium on Operating Systems Principles* (SOSP 2003), New York, USA, October, 2003.

[7] H. Hardware, and P. Across, "The Hadoop Distributed File System: Architecture and Design", 2007, pp. 1–14.

[8] H.-P. Chang, R.-S. Chang, and Y.-T. Wang, "A dynamic weighted data replication strategy in data grids", *2008 IEEE/ACS International Conference on Computer Systems and Applications*, Mar. 2008, pp. 414–421.

[9] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling", *In Proceeding of uropean Conference Computer System* (EuroSys), 2010.

[10] https://webscope.sandbox.yahoo.com.

[11] Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall, 1992.