Data Popularity-Aware Replication Strategy for Cloud Storage

MAY PHYO THU[1], KHINE MOE NEW[2], KYAR NYO AYE[3]
[1, 2]Faculty of Computer Science, University of Computer Studies, Yangon, Myanmar

Abstract- Replication is one of the important roles in cloud storage to improve data availability, fault tolerance and throughput for users and control storage cost. As data access pattern changes every time, the nature of popular files is unpredictable and unstable. Therefore, data popularity is taken into account as an important factor in replication. Data popularity in replication impacts an efficient storage because it is able to reduce waste storage for unpopular files. Also, data locality is a key issue in storage system and this consequence occurs performance overhead of system. Therefore, this paper introduces a replication strategy for cloud storage. The proposed strategy contains two portions; replica popularity and replica placement. First for replica popularity, popularity is taken into account by analyzing the changes in data access pattern. Second for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes such as disk utilization, adjustable disk bandwidth and CPU utilization. This proposed strategy will be efficient for cloud storage.

Indexed Terms- Popularity, Data Locality, Disk Utilization, Adjustable Disk Bandwidth, CPU Utilization

I. INTRODUCTION

Users are allowed to saves their files and given access permissions to them through cloud in cloud storage technology. Cloud storage is one of the services provided by cloud computing. In cloud storage, data among geographically distributed multiple servers is converged into a single place and users is provided with immediate access to this data for cloud-based applications. It consists of a cluster of storage nodes or even geographically distributed data centers. There are many cloud storage products such as Google File System (GFS) [7], Simple Storage Service (S3), Hadoop Distributed File System (HDFS) [8] etc. Among them, HDFS provides reliable storage for very large data sets and streams at high throughput access to these data sets. In HDFS, data is divided into defined -size blocks and then these data blocks are placed at data nodes with replication.

In Hadoop, when users submit a job, the incoming job is executed on these data, such that Hadoop splits each job into tasks and then assigns each mapper with these data blocks. In this case, if there is no data block for this mapper at assigned node, these data block is have to be copied from hosted node to that assigned node in that the mapper task is executed. In order to provide data locality, Hadoop attempts the collocation of data with assigned node. Data locality is one of key issues in Hadoop. Better data locality provides the minimal of network congestion and the increment of overall system's throughput. Three types of data locality are node locality, rack locality and rack-off locality. Current implementations of Hadoop use uniform data replication. In data replication, file popularity that represents whether a file has been hot in recent time intervals and is calculated by analyzing file access rate.

In this system, data replication strategy based on data popularity is proposed in order to provide efficient replication strategy for cloud storage. In replica allocation, the analysis of file access patterns using differential equation is performed to predict the increment and decrement of file popularity and then, the number of replicas for each file is computed according to the result of the prediction. In replica placement, allocation of replicas is processed using proposed data placement algorithm to provide better data locality.

The contributions of this system are as follows:

1) Changes of file popularity in timeslots is analyzed using first order differential equation.
2) Increment and decrement of the number of replicas for each file is computed.
3) While the replicas are placed into nodes, the load of nodes such as disk utilization, CPU utilization and bandwidth utilization are considered.
4) The predefined threshold is used to compute the overloaded condition of cluster.
5) If the overloaded condition of that assigned nodes occurs, proposed replica replacement algorithm is used.
6) This proposed replacement algorithm considers not only outgoing blocks but also the access frequencies for blocks.

The rest organizations of paper are: related works are described in section 2. Section 3 presents background theory and proposed system architecture is shown in section 4 and performance is evaluated in section 5 and the conclusion and future work is concluded in section 6.

II. RELATED WORKS

Cloud storage provides a storage services that is hosted remotely on servers and users can access this through Internet. Data is replicated and maintained in several nodes to provide high availability and load balancing. There were several researches on data replication in cloud storage. In [2], the authors proposed a replication scheme that focused to achieve file availability by resolving least replicas for a file, and to assign them at nodes for attaining in performance improvement and load balancing. In data placement, blocking probability is considered as a factor in order to eliminate access skew and improve concurrency. Nevertheless, it wasn't good for terabyte-sized file. R.S. Chang and H.P. Chang proposed an algorithm for data grids, Latest Access Largest Weight (LALW) [9]. It detected only the most popular file in each timeslot and computed replicas for that popular file and determined which nodes were suitable to place these replicas. This did not consider unpopular files and so did not eliminate unnecessary replicas.

A. Hunger and J. Myint introduced a replication algorithm that is based on file popularity: Pop Store [1]. In that paper, they detected more popular files at timeslots using Half-life approach. However, file popularity does not have always decay over the time because the characteristics of data access is dynamic. This paper did not consider this condition and data locality in replica placement. There have been many researches concerning with improving data locality on data replication in Hadoop. Scarlett [6] presented a replication method that replicated proactively files according to prediction of data popularity. It targeted to data received at least three requests concurrently. However, they did not think about node popularity occurred by relative popular data arrangement.

For achieving improved data locality, the authors proposed a distributed adaptive data replication algorithm (DARE) based on the access frequencies of data blocks [3]. It was a reactive approach and it retained remote data retrieval and evicted aged replicas. It automatically increased the number of replicas when data was replicated to the fetched node. However, it had the limitation of the optimized number of replicas. To find the solution of dispute between data locality and equity on jobs, a delay scheduling algorithm was introduced in [10]. Delay scheduling allowed jobs to wait for a small amount of time, as a consequence, it disobeyed the equity of jobs. In that, it took assumptions that task durations were short and bimodal, and a fixed waiting time worked for all loads and skewness of traffic. Therefore, it was not adaptable to changes in workload, node popularity or network conditions.

Also, access count prediction-based data replication scheme for Hadoop was proposed in [4]. This scheme determined whether generation of a new replica or use of data as cache selectively using the predicted data access count. It placed replicas into nodes using the structure of circular linked list without consideration of utilization factor of this nodes. In [5], the authors introduced adaptive replication method for supporting availability by increment of data locality, as a consequence, it increases the performance of Hadoop. It used supervised learning for prediction of file access, determination of replicas and placement of replicas. A proactive re-replication scheme based on predicted CPU and disk utilization was proposed as keeping balance condition of nodes. It applied local regression with historical information of each node to predict CPU and disk utilization. That obtained utilization information was used to perform an efficient re-replication scheme. It used priority-based grouping to obtain fairness between reliability and performance.

III. BACKGROUND THEORY

Replication is widely used in storage systems to enhance the efficiency of data access and the fault-tolerance. Data locality is considered as a principal issue in Hadoop. This issue occurs when the computing node performs remote data retrieval as there has no assigned replica block to be processed. The proposed replication strategy takes into account the data popularity while determination of replicas and data locality when allocation of replicas. This section describes architecture of Hadoop cluster and data locality.

3.1 Architecture of Hadoop Cluster

Hadoop is open-sourced, platform-independent and it provides faster access of data in distributed applications. The structure of Hadoop has two parts: MapReduce and HDFS. MapReduce keeps user jobs and tasks and HDFS has responsibility of data storage, data blocks management and their metadata' information. It keeps three replicas as like GFS.

A job tracker performs division of entering job into many tasks and assignment of tasks with task trackers. In order to detect the status of task trackers, the job tracker performs the collection of heartbeat information send by task trackers. When data are stored in HDFS, breaking of data into fixed-sized blocks with replication and these blocks are stored in slave nodes. The task tracker is responsible for arrangement of tasks in the node. When there is no task slot, it sends a heartbeat information to job tracker in order to demand a task.

Splitting of input data into blocks and placement of blocks at nodes are performed when the user maintains data in them. The job tracker has responsibility for handling mapreduce job requests of client. When the job tracker receives the request, it split a job into tasks and allocates these tasks with task trackers in consideration of data locality. Then, allocation of task with node is performed by each task tracker and that assigned node performs the task with pulling block from HDFS if necessary.

When users make submission of MapReduce jobs, Hadoop performs splitting of job into tasks. Then, input data is broken into predefined-size blocks separately and map tasks executes them in parallel and distribution among nodes in the cluster. Each input block has one map task. After execution of map tasks, the obtained output is shuffled, sorted and performed in parallel by one or more reduce tasks.

3.2 Data Locality

At Hadoop framework, data storage is performed in HDFS. It breaks down the input data into predefined-size blocks and these blocks are allocated at nodes in cluster. Each mapper operates the blocks when a job is operated on the dataset. If the condition, that is lack of replica at computing node for map task will occur, prefetching needed replica block into this node. At this condition, data locality problem occurs.

Data locality has relations with the interval between data blocks and the computing node. Data locality is interval between data block and the assigned node. The closer the distance, the greater data locality. The greater locality, the more throughput of the system. Types of data locality are:

1) Node locality: operation data is kept at assigned node
2) Rack locality: operation data is not kept at assigned node, however at another node within one rack,
3) Rack-off locality: operation data is kept at other node in separate rack. Among these types of data locality, the most preferred scenario is node locality and the least preferred scenario is rack-off locality.

This locality problem occurs when the colocation of task with node is not in the same place. Moreover, the impact of rack-off locality is worst in them. In order to eliminate the impacts of the data locality problems, we propose a replication strategy using prediction of the file access count and a data placement algorithm in decreasing condition of rack and rack-off locality.

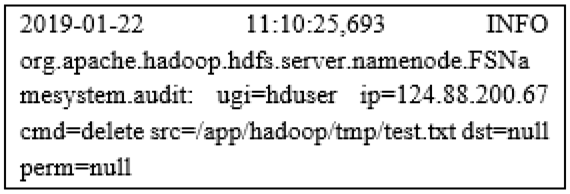
IV. PROPOSED SYSTEM ARCHITECTURE

The basic idea of replication is based on different replication degree per data file. Keeping the fixed number of replicas causes wasteful storage for unpopular data and inefficiency for popular data. Also, maintaining too much replicas than current access count for a file does not always guarantee better locality for all blocks. The proposed system flow diagram is presented in figure 2. The aim is to develop a replication technique attempted for improvement of data locality by increasing replicas for popular data while maintaining less replicas for unpopular data.

Firstly, we calculate changes of file popularity with first order differential equation. The assumption of popularity is that the popularity of an item grows at a definite time is relative to the total popularity of the item at that time. From the first order differential equation, we compute the growth or decay constant, k.

$$k = \frac{\ln(\frac{P(t)}{P_0})}{t} \quad (1)$$

where P(t) denotes popularity at time t, P0 is starting popularity and k is growth or decay constant. From Yahoo Hadoop audit log file data source [11], we compute changes of file popularity. The log file is divided into smaller files according to timeslot duration. After that, extraction of fields such as Date, Time and src is performed. Then, from the src link, access frequency is counted in each timeslot. Then, changes of file popularity, k, is computed with above mentioned equation 1. The Yahoo HDFS User Audit log format is presented in figure 1.



```
2019-01-22          11:10:25,693          INFO
org.apache.hadoop.hdfs.server.namenode.FSNa
mesystem.audit:  ugi=hduser  ip=124.88.200.67
cmd=delete src=/app/hadoop/tmp/test.txt dst=null
perm=null
```

Figure. 1: HDFS user audit log format

At second stage, the number of replicas for each file is defined using changes of file popularity that is the outcome of the first stage. Initially, existing replicas will be assumed as 3 as like the default replica of HDFS. If k is less than 0.0, then existing replicas is decreased by 1. If k is greater than 0.0, then existing replicas is increased by 1. If k is equal to 0.0, then existing replicas is unvaried. Otherwise, if it is new file, then existing replicas is determined 3 as like the default replica of HDFS.

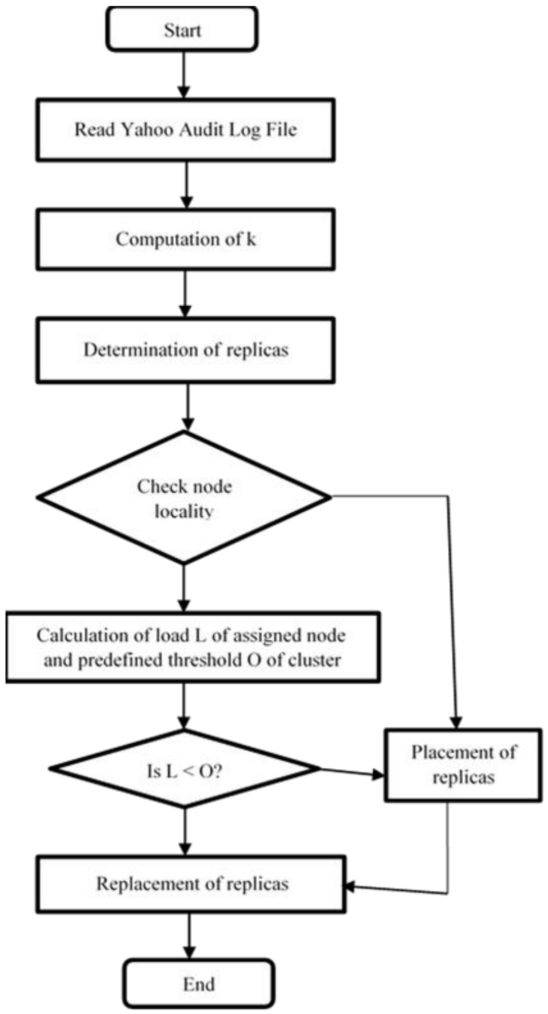

Start
Read Yahoo Audit Log File
Computation of k
Determination of replicas
Check node locality
Calculation of load L of assigned node and predefined threshold O of cluster
Is L < O?
Placement of replicas
Replacement of replicas
End

Figure. 2: Proposed System Flow Diagram

At the third step, replicas are placed into assigned nodes to achieve greater data locality. We will make the assumption that the incoming jobs must have to access these replicas at next timeslot.

The entering job is split into tasks and assignment of task with nodes in the cluster is performed. Each input block has one map task. It is assumed that one data block represents one data file. We will let that maximum replicas are total nodes in the cluster and minimum replicas is 1. Node locality of task is checked and if there has node locality, then placement of task at that assigned node is performed. If the condition, that is lack of replica at computing node for map task will occur, prefetching needed replica block into this node. In this system, the load of assigned node is considered to avoid overloaded condition while loading into assigned nodes. That replica is loaded if the load of assigned node is less than predefined threshold. Otherwise, replacement of needed replica block with existing block at assigned node is performed.

The default placement policy of Hadoop is randomness and it assumes that all nodes within cluster have equality condition. Moreover, it does not consider utilization of nodes in placement. This condition results in imbalance load to Hadoop. The proposed system considers inequality condition of nodes within the cluster. In this system, we consider disk utilization, adjustable bandwidth and CPU utilization as the load of nodes. We can carry out the disk utilization as

$$U(D_i) = \frac{D_i(use)}{D_i(total)} \quad (2)$$

Where, U(Di) is the disk utilization of the ith node, D(i) (use) is the used disk capacity of the ith node and D_i (total) is the total disk capacity of the ith node. Then, we can carry out the disk bandwidth as

$$BW(D_i) = \frac{T_b}{T_s} \quad (3)$$

Where, BW(D_i) is the disk bandwidth of the ith node, 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 adjustable disk bandwidth of node for load factor is considered as

$$ABW(D_i) = \frac{BW(D_i)}{Total_i(BW)} \quad (4)$$

Where, $ABW(Di)$ is the adjustable bandwidth of the ith node and $Totali$ (BW) is the total bandwidth of the ith cluster. We can carry out the CPU utilization as

$$CU(D_i) = 100\% - (\text{Percentage of time that is spent in idle task}) \quad (5)$$

Where, CU(D_i) is the CPU utilization of the ith node. To compute the load of assigned node, the coefficients of storage utilization, disk bandwidth and CPU utilization are set as $\propto$, β and γ. Then, we can carry out the load of node as

$$Load(D_i) = \propto U(D_i) + \beta\, BW(D_i) + \gamma\, CU(D_i) \quad (6)$$

That replica is placed at that node if the load of assigned node is less than predefined threshold Ti. Otherwise, replacement of needed replica block with existing block at assigned node is performed. The proposed data replacement algorithm is based on Least Recently Used (LRU). It is more reliable and efficient than LRU because it takes into account not only outgoing blocks but also access frequencies for blocks in replacement. The proposed data replacement algorithm and data placement algorithm are as follows:

Algorithm 1: Data Replacement Algorithm
Step 1: It compute total access frequencies of all blocks at that assigned node as the replica is loaded into the assigned node.

Step 2: That replica is selected to evict from the node if only one block that has minimum access frequencies is found.

Step 3: If there have more than one block that have minimum access frequencies are found, outgoing block is chosen to remove from that assigned node as LRU.

Figure. 3: Data Replacement Algorithm

Table 1: Notations Used in Data Placement Algorithm

Notation	Description
D	Nodes list
ABW	Adjustable bandwidth
U	Disk utilization
RE	Replica List
MAP	Map task list
CU	CPU utilization
CL	Cluster list
L	Load factor list

```
Algorithm 2: Data Placement Algorithm
Input: Nodes List D= {D1, D2 ,.., Dn }, Replica List RE ={ RE1, RE2, RE3,…., REn }, Map Task List MAP = {MAP1, MAP 2, MAP 3,…, MAP n}, Load Factor List L = {L1,L2,L3,…., Ln}, Predefined Threshold Ti, Cluster List C = {C1, C2, C3,…., Cn}
Output: Nodes List D
 for each entering map task MAP do
   for each node D do
     Detect node locality of task MAPi
     if it has node locality then allocate task MAPi to that node Di
     else
       Process remote data retrieval for task MAPi
       Calculate disk utilization U of this assigned node Di using (2)
       Calculate adjustable disk bandwidth ABW of this assigned node Di using (4)
       Calculate CPU utilization CU of this assigned node Di using (5)
       Calculate load factor Li for this assigned node Di using (6)
       Calculate predefined threshold Ti for the cluster CLi
       if Li > predefined threshold Ti then
         Perform replacement using algorithm 1
         Place replica REi for this task on that node Di
         break
       else
         Place replica REi for this task on that node Di
         break
       end if
     end if
   end for
 end for
```

Figure 4. Data Placement Algorithm

V. PERFORMANCE EVALUATION

The replication algorithms are implemented and tested. The experiments are set up by using one evaluation parameter: disk utilization. In this proposed system, the replicas are almost uniformly distributed for achieving the load balancing in nodes in the cluster. Disk utilization of the proposed system are compared with LALW algorithm in order to avoid overload condition. LALW does not obey the placement policy of hadoop because it places the same data replicas at one host. Therefore, LALW does not achieve the load balancing as like proposed system.

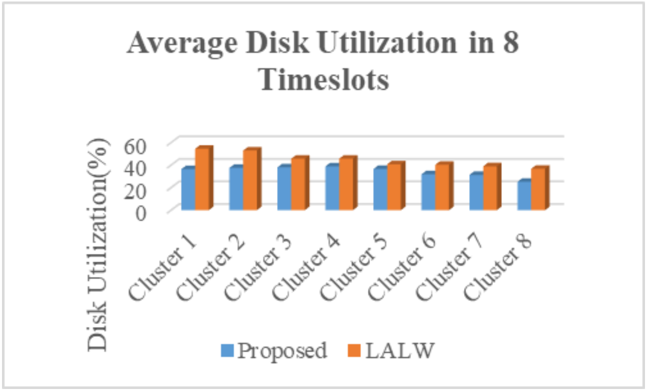


Figure 5. Average Disk Utilization of Proposed System and LALW

VI. CONCLUSION

Cloud storage provides a storage services that is hosted remotely on servers and users can access this through Internet. Data is replicated and stored in multiple data nodes to provide for data availability. This paper focuses on changes of data access pattern due to unpredictable popularity. The allocation of unpopular data leads to waste in cloud storage. The proposed replication strategy will overcome this issue of waste in cloud storage. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes such as disk utilization, adjustable disk bandwidth and CPU utilization. This system provides optimum replica number as well as enhancing data locality and load balancing among the storage server nodes. Performance evaluation such as disk utilization is compared with LALW algorithm. According to the experimental results, this proposed system is more load balancing than LALW. And as well, this data replication scheme will be implemented in various distributed file systems as an ongoing research.

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] D.M. Bui, S. Hussain, E.N. Huh, and S. Lee, “Adaptive replication managementin hdfs based on supervised learning,” IEEE Transcations on Knowledage and Data Engineering, vol.28, no.6, 2016.

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

[7] 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.

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

[9] 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.

[10] 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.

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

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