Dynamic Replication Management Scheme for Distributed File System

May Phyo Thu, Khine Moe Nwe(✉), and Kyar Nyo Aye(✉)

Faculty of Computer Science, University of Computer Studies (Yangon), Yangon, Myanmar
mayphyothu.mptl@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 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].

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., 32 MB, 64 MB, and 128 MB) 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. Data locality is a principal factor of Hadoop's performance. Data locality means the degree of distance between data and the processing node for the data. There are three types of data locality in Hadoop: node locality, rack locality

T. T. Zin and J. C.-W. Lin (Eds.): ICBDL 2018, AISC 744, pp. 139–148, 2019.
https://doi.org/10.1007/978-981-13-0869-7_16

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. 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 proposed system architecture is presented in Sect. 3. Section 4 presents performance evaluation and finally, Sect. 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. A cost effective replication management scheme for cloud storage cluster was proposed by 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. One approach, Latest Access Largest Weight (LALW) algorithm [7], that was proposed by Chang and 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.

Hunger and 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. 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. 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 [8], the authors proposed a delay scheduling method that focused on the conflict between data locality and fairness among jobs. However, delay scheduling made assumptions that might not hold universally and 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. Although this scheme was designed to improve data locality, it considered file level replication did not consider block level replication.

3 Proposed System Architecture

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 Fig. 1.

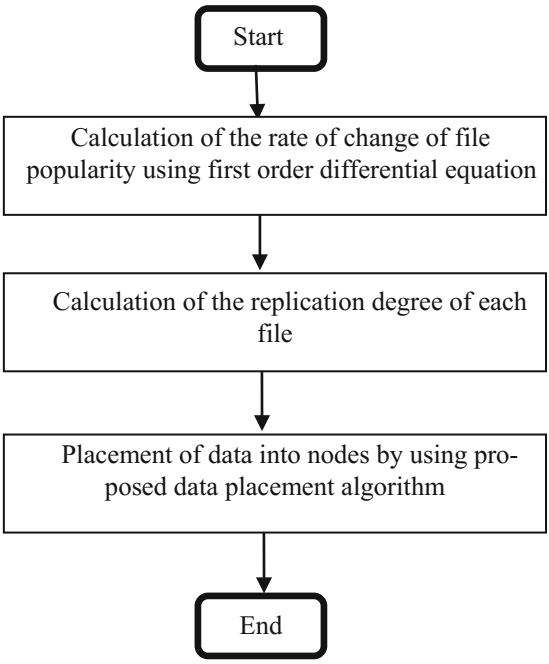

Fig. 1. Proposed system flow diagram

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.

3.1 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

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

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

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

Then,

$$k = \frac{\ln\left(\frac{P(t)}{P_0}\right)}{t} \tag{3}$$

Where P_0 is the initial population, i.e. p (0) = P0, 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 [9] as data source. Users can enable audit logging from the NameNode. The Yahoo HDFS User Audit log format is shown in Fig. 2.

2016-12-1011:11:59,693INFO 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

Fig. 2. 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 Fig. 2. 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 1 and Fig. 3.

Table 1. 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 $P(t)$ as $P(t_f)$ in (3)
4. return k.

Fig. 3. Popularity growth rate algorithm

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, then existing replica degree is increased by one. If the rate of change of file popularity is less than 0, then existing replica degree is decreased by one. If the rate of change of file popularity is equal to 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.

3.2 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. The proposed enhanced LRU replacement algorithm is shown in Fig. 4.

The existing Hadoop block placement strategy does not take into account Data-Nodes' utilization, which leads to in an imbalanced load. This policy assumes that all nodes in the cluster are homogeneous, and randomly place blocks without considering

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.

Fig. 4. Enhanced LRU algorithm

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

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

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

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

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 $\propto$, β and γ. Therefore, we can carry out the load factor model as

$$LF(D_i) = \alpha U(D_i) + \beta BW(D_i) + \gamma SP(D_i) \tag{6}$$

The predefined threshold Ti of the ith 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 Ci as

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

Where, T_i is the predefined threshold of the ith cluster and N is the number of nodes in the ith cluster. If the load factor of this node is less than the predefined threshold, this replica data block is loaded into this node. The proposed data placement algorithm is shown in Table 2 and Fig. 5.

Table 2. 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

4 Performance Evaluation

In this section, the probabilistic model is applied to evaluate the analysis of availability. In this model, data file F is stripped into n blocks denoted as B = {b1,b2,b3,…,bn} and stored into different DataNodes. There are rn replicas of each block in data file F, and all blocks at the same site will have the same available probability as all blocks are stored in DataNodes with the same configuration. Assume that NameNode do not fail any condition. To analyze the data availability of this system, assume that node failures are independent with failure probability. If a node fails, all of the blocks will lose on the nodes. The block availability of data block Bn is denoted as BA_n. $P(BA_n)$ is the probability of data block Bn in an available state. $P(\overline{BA_n})$ is the probability of data block Bn in an available state. Let the number of replicas of data block bn be r_n and the available and unavailable probability of each replica of data block bn are $P(b_n)$ and $P(\overline{b_n}) = 1 - P(b_n)$. So, the availability and unavailability of data block Bn are calculated as

$$P(BA_n) = 1 - (1 - p(b_n))^{r_n} \tag{8}$$

$$P(\overline{BA_n}) = (1 - p(b_n))^{r_n} \tag{9}$$

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
```

Fig. 5. Data placement algorithm

Consider the probability of each replica block bn in the system of n DataNodes that containing b data blocks replicated with the replication factor of br. So, the probability of bn available is

$$p(b_n) = \sum_{\mathrm{f}=0}^{\mathrm{n}} p(failure) \times p(\text{no - loss}) \tag{10}$$

where $p(failure)$ is the probability that there are exactly failures in the system with n DataNodes. So, $p(failure)$ is

$$p(failure) = \binom{n}{f} \times p^f \times (1-p)^{(n-f)} \tag{11}$$

$p(\text{no\_loss})$ is the probability of not losing data in the system with f failures. To calculate $p(\text{no\_loss})$, firstly we compute the probability of losing a block of data when there is f failure that is defined as $p\left(\overline{single_{block}}\right)$. The two are related as

$$p(no\_loss) = 1 - p\left(\overline{single_{block}}\right)^{b} \tag{12}$$

The probability of losing a block of data when there is f failure in n DataNodes is

$$p\left(\overline{single_{block}}\right) = \binom{f}{br} / \binom{n}{br} \tag{13}$$

Therefore, we get

$$p(b_n) = \sum_{f=0}^{n} \binom{n}{f} \times p^{f} \times (1-p)^{(n-f)} \times \left(1 - \binom{f}{r} / \binom{n}{r}\right)^{b} \tag{14}$$

To retrieve the data file F, we need to get all block $\{b_1, b_2, b_3, \ldots, b_n\}$. Any block unavailable will cause file unavailable. The file availability of file f_n is denoted as FA_n. $P(FA_n)$ is the probability of file f_n in an available state. $P\left(\overline{FA_n}\right)$ is the probability of file f_n in unavailable state. The availability and unavailability of file f_n are $P(FA_n)$ and $P\left(\overline{FA_n}\right) = (1 - P(FA_n))$. Then, the availability and unavailability of file f_n are calculated as

$$P(FA_n) = (1 - (1 - p(b_n)^{r_n})^{b_n} \tag{15}$$

$$P\left(\overline{FA_n}\right) = 1 - (1 - (1 - p(b_n)^{r_n})^{b_n} \tag{16}$$

To evaluate the availability of data block, the 1 GB of data is stored in Cluster based storage system. The average failure rate of each DataNode in the system is 0.01, 0.02, 0.03, 0.04 and 0.05 (Fig. 6).

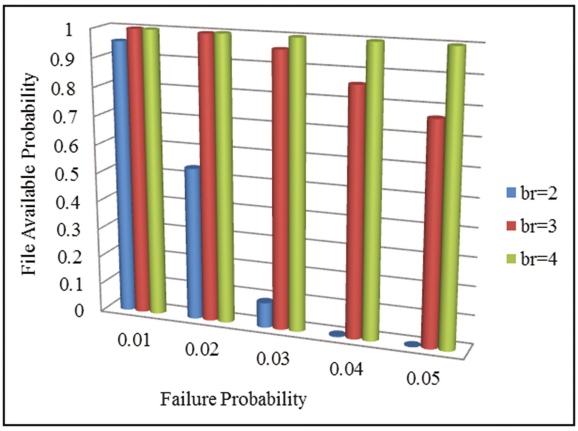

Fig. 6. File availability with replication factor (r) 2, 3 and 4 in the eighty DataNodes

It can be observed from the evaluation results, and file availability is a rapidly fall on replication factor 2 in the eighty number of data nodes in the system. According to the evaluation results, the file availability depends on the replication factors and failure probability. From the experiment, it is obvious that the expected availability values can be satisfied by increasing replication degree in the existence of failure on cluster.

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. 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, the rate of change of file 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.

References

1. Hunger, A., Myint, J.: Comparative analysis of adaptive file replication algorithms for cloud data storage. In: 2014 International Conference on Future Internet of Things and Cloud (2014)
2. Gong, B., Veeravalli, B., Feng, D., Zeng, L., Wei, Q.: CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster. In: 2010 IEEE International Conference on Cluster Computing, September 2010, pp. 188–196 (2010)
3. Abad, C.L., Lu, Y., Campbell, R.H.: DARE: adaptive data replication for efficient cluster scheduling. In: IEEE International Conference on Cluster Computing (CLUSTER 2011), pp. 159–168 (2011)
4. Lee, D., Lee, J., Chung, J.: Efficient data replication scheme based on hadoop distributed file system. Int. J. Softw. Eng. Appl. **9**(12), 177–186 (2015)
5. Ananthanarayanan, G., et al.: Scarlett: coping with skewed content popularity in mapreduce clusters. In: Proceedings of Conference on Computer. Systems (EuroSys), pp. 287–300 (2011)
6. Gobioff, H., Ghemawat, S., Leung, S.-T.: The Google file system. In: Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP 2003), New York, USA, October 2003
7. Chang, H.-P., Chang, R.-S., Wang, Y.-T.: A dynamic weighted data replication strategy in data grids. In: 2008 IEEE/ACS International Conference on Computer Systems and Applications, March 2008, pp. 414–421 (2008)
8. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceeding of European Conference Computer System (EuroSys) (2010)
9. https://webscope.sandbox.yahoo.com