RTRM: A Response Time-Based Replica Management Strategy for Cloud Storage System

Xiaohu Bai, Hai Jin, Xiaofei Liao, Xuanhua Shi, and Zhiyuan Shao

Services Computing Technology and System Lab.
Cluster and Grid Computing Lab.
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract. Replica management has become a hot research topic in storage systems. This paper presents a dynamic replica management strategy based on response time, named RTRM. RTRM strategy consists of replica creation, replica selection, and replica placement mechanisms. RTRM sets a threshold for response time, if the response time is longer than the threshold, RTRM will increase the number of replicas and create new replica. When a new request comes, RTRM will predict the bandwidth among the replica servers, and make the replica selection accordingly. The replica placement refers to search new replica placement location, and it is a NP-hard problem. Based on graph theory, this paper proposes a reduction algorithm to solve this problem. The simulation results show that RTRM strategy performs better than the five built-in replica management strategies in terms of network utilization and service response time.

Keywords: Dynamic replica management, Response time, OptorSim, Load balance.

1 Introduction

Since data replication has been widely used in storage systems [1-3], replica management has been a hot research topic [4-9]. As the storage environment changes dynamically, dynamic replica management gets more attention by researchers. Replica management includes replica creation, selection, and placement.

Most existing dynamic replica management strategies create new replica of the popular data based on the user access frequency, thus the replica creation always happens at the end of each time interval. But according to temporal locality and spatial locality, especially the pattern of user accesses, the distribution of the user accesses is uneven during the time interval. A file may have many concurrent requests during the time interval, and these concurrent requests will greatly increase the service response time of each single request. Two issues should be addressed: (1) when is the best time for replica creation of popular data to reduce the average service response time; (2) how many replicas can satisfy the response time requirement of a single request.

In this paper, we focus on the response time of a single request, and propose a response time-based replica management strategy, named RTRM, which includes three algorithms: replica creation, replica selection, and replica placement. Replica creation algorithm decides when and where to create replica based on the average response time. Replica selection method selects the best replica node for users based on response time prediction, while replica placement mechanism combines the number of replicas and the network transfer time. To evaluate the performance of RTRM, we run the strategies in OptorSim [10]. The evaluation results show that our replica management strategy performs better than the five built-in replica management strategies in OptorSim simulator in terms of service response time and network utilization.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents dynamic replica management strategy. The analysis and evaluation results are presented in section 4. In section 5, we give conclusions and possible future work.

2 Related Works

Replica management has been widely studied. Sun et al. [4] proposed a replica strategy based on the memory cache. Hou et al. [5] proposed a dynamic replica creation mechanism DynRM, which decides to create replicas according to the file access frequency. Chang et al. [6] set access-weights for each file, and choose hot file based on the value of access-weights. These replica strategies do not take the response time of a single request into consideration, while many requests have to be waiting for a long time.

Rahman et al. [7] proposed a replica placement algorithm used the p-median model to find the locations of p candidate nodes to place replicas, but the problem is how to determine an appropriate value of p. A model-driven replica strategy is proposed in [8]. This strategy first calculates the requisite number of replicas and selects the best set of nodes to host the replicas. However, as each node can only utilize partial information, this strategy may create too many replicas and result in prohibitive overhead. Li et al [9] proposed a DSRL replica location method in which each file has a home node to maintain the index of all the replicas. With the dynamic changes in the network, DSRL method would create too many replicas.

3 Design of RTRM

3.1 Replica Creation Method

In dynamic replica management strategy, replica creation decides which file is the popular data and when is the right time to create new replica of the popular data. Replica creation method first finds the best time to create new replica, an access recorder is assigned to each data node, which is used to store the number of concurrent user accesses to each file, including file name, number of concurrent access, file size, and so on. The service response time of single access can be calculated by the number of concurrent user accesses. Once the average service response time of a file is higher than a threshold, the file becomes popular data, and the creation of that file is started.

In our replica creation method, $T_{threshold}$ is set as the upper limit of the service response time of a single request. The average service response time of a file must be smaller than $T_{threshold}$.

Assume that data block b has n replicas, and distributed in n nodes. Let these n nodes be N_1, N_2, … N_n. To simplify the problem, for the user accesses of data block b, we have the denotations as follows:

The size of data block b is denoted as S_b.

The network transmission capability of node N_i is denoted as NTC_i.

The number of concurrent accesses of node N_i is denoted as Num_i.

The maximum service response time of single request of node S_i is denoted as $MSRT_i$. $MSRT_i$ can be computed by Equation (1).

$$MSRT_i = \frac{S_b}{NTC_i} \times Num_i (i = 1, 2, ..., n) \tag{1}$$

We define $MSRT_{MAX}$ as the maximum value of all $MSRT_i$, the average response time of all $MSRT_i$ is denoted as $MSRT_{average}$. Based on Equation (1), $MSRT_{MAX}$ and $MSRT_{average}$ can be computed by Equation (2).

$$\begin{cases} MSRT_{MAX} = \max(MSRT_1, MSRT_2, ..., MSRT_n) \\ MSRT_{average} = \frac{1}{n}\sum_{i=1}^{n} MSRT_i \end{cases} \tag{2}$$

Each time when a user access comes, we get the value of $MSRT_{MAX}$ and $MSRT_{average}$ through Equation (2). If the value of $MSRT_{average}$ is higher than $T_{threshold}$, file f is considered to be popular data, and new replica of file f will be created. If $MSRT_{average}$ is smaller than $T_{threshold}$, but $MSRT_{MAX}$ is higher than $T_{threshold}$, then the system would transfer some accesses from the relatively heavy load nodes to the relatively light load nodes.

3.2 Replica Selection Method

The goal of replica selection method is to select the best replica node of a file. In replica selection method, *LPC* is defined to represent the load process capability of a node. The metrics of *LPC* consists of three components: CPU process capability, network transmission capability, and I/O capability of disks, denoted by w_c, w_n, w_{io}, respectively. Given these metrics, *LPC* can be computed by Equation (3).

$$LPC = \alpha * w_c + \beta * w_n + \gamma * w_{io} \tag{3}$$

In Equation (3), α, β, γ are constants and can be determined according to service level. Replica selection method chooses the node with highest *LPC* to response the user request, the user then accesses the file from the node with highest *LPC*.

3.3 Replica Placement Mechanism

Replica placement has been proven to be NP-hard. We first give a model of replica placement, and then we propose a reduction algorithm to solve this problem.

Assume that the system has n storage nodes, let them be n_1, n_2, …,n_n. We want to get the minimal replicas of file f, and place these replicas to satisfy the requirement of a single request. To simplify the problem, the denotations are as follows:

(1) The replica number is denoted as *replicaDegree*, and the upper limit of the response time of a single request is set as T_{upper}.

(2) The response time that node n_i accesses file f is denoted as $responseTime_i$, it is the time that n_i accesses file f from the nearest node. If n_i contains file f or its replica, $responseTime_i$ is set to be 0.

(3) The total response time of the system is denoted as *TotalresponseTime*, and *TotalresponseTime* can be computed by Equation (4).

$$TotalresponseTime = \sum_{i=1}^{n} responseTime_i \tag{4}$$

The goal of our design is to make sure that the response time of a single request must be smaller than T_{upper}, and minimize the value of *replicaDegree* and the value of *TotalresponseTime*. Therefore, in this paper, we want to find an optimal replica scheme that can achieve the following goals:

(1) Minimize *replicaDegree*
(2) $responseTime_i <= T_{upper}$
(3) Minimize *TotalresponseTime.*

For goals (1) and (2), they can be described as a *Set Covering Problem* (SCP), which has been proven to be NP-hard. Based on greedy algorithm, by transforming the SCP into an equivalent graph, we design a reduction algorithm to figure out this model.

Based on the network topology and the network transfer time, we construct a graph $G=(V, E)$, this graph can be described as:

$V=\{n_1, n_2, …, n_n\}$; $E=\{(n_i, n_j) \mid responseTime_{ji} <= T_{upper}\}$.

As an example, a network topology and the network transfer time is shown in Fig.1, and the value of T_{upper} in this example is 10s.

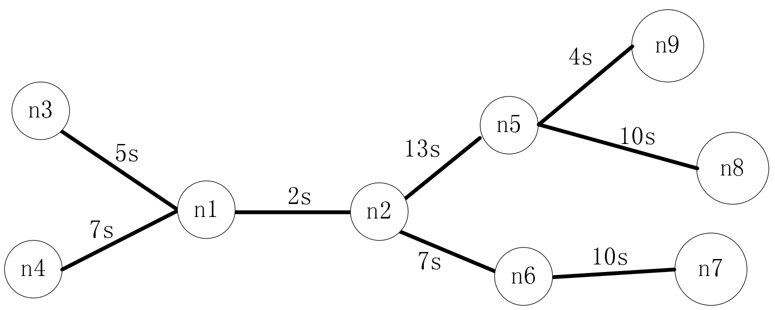

Fig. 1. Network topology and network transfer time

From the graph, we can get the value of V and E.

$V=\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$; $E=\{(n_1, n_2), (n_1, n_3), (n_1, n_4), (n_1, n_6), (n_2, n_3), (n_2, n_4), (n_2, n_6), (n_5, n_9), (n_5, n_8), (n_6, n_7)\}$.

The goal is to find a subset V^*, which is a smallest subset of V, for each element v from V, there must have at least one element v^* from V^*, and (v, v^*) is an element in E. It means that for each node v in V, there must be at least one node v^* in V^*, and v can access file from v^* within T_{upper}.

Algorithm 1 shows the process of the reduction algorithm. We can place the replicas in the nodes from V^* to make sure that all the nodes can access file f within T_{upper}.

Algorithm 1. Reduction algorithm

```
INPUT: $G = (V, E)$; OUTPUT: $V^*$
// degree(v) gets the degree of v in G;
1. Begin
2.          Initialize $V^*$ and $v^*$: $V^* = \emptyset$, $degree(v^*) = 0$;
3.          if ( $V == \emptyset$ ) {go to 18;}
4.          else {go to 5;}
5.          for ( each element v in V )
6.                   if( $degree(v) > degree(v^*)$ ) { $v^* = v$;}
7.                   push $v^*$ into $V^*$;
8.                   delete all the edges incident to $v^*$ from $V$;
9.                   delete $v^*$ from $V$;
10.         end for
11.         if ( $V == \emptyset$ ) {go to 18;}
12.         else {go to 13;}
13.         for ( each element v in V )
14.                  if ( $(v^*, v) \subseteq E$ )
15.                  {if($degree(v) == 0$) { delete $v$ from $V$;}}
16.         end for
17.         go to 3.
18.         return $V^*$;
19. End
```

4 Performance Evaluation

In this section, we first compare our replica placement mechanism with other four replica placement strategies, then compare RTRM strategy with the five built-in replica strategies in OptorSim. From the experiment results, RTRM strategy performs better in terms of network utilization, average response time, and total replica number.

4.1 Analysis of Replica Placement Mechanism

We will compare our replica placement mechanism with other four strategies: *Best Client*, *MinimizeExpectedUtil*, *MaximizeTimeDiffUtil*, and *MinimizeMaxRisk.*

The example in Fig. 1 is used in the analysis. The upper limit of the response time of a single request T_{upper} is set to 10s. We define *replicaDegree* to represent the number of replicas in the system, and use *TotalresponseTime* to represent the total response time of all nodes in the system. We perform two analyses. In the first analysis, we compare the value of *TotalresponseTime* of the five mechanisms with the same *replicaDegree*. In second analysis, we compare the smallest *replicaDegree* of the five mechanisms while making sure the response time of all requests is smaller than T_{upper}.

First Analysis

Because in general storage systems, the smallest replica degree is 3, we set the value of *replicaDegree* of all the five mechanisms 3, and access the file from each node, then compare the *TotalresponseTim*e of each mechanism. Result is in Table 1.

Table 1. Results of first analysis

Mechanism	*TotalresponseTime*	Nodes to host replica
RTRM	40	n_2, n_5, n_6
Best Client	77	n_2, n_3, n_4
MinimizeExpectedUtil	48	n_1, n_2, n_5
MaximizeTimeDiffUtil	52	n_1, n_2, n_9
MinimizeMaxRisk	69	n_2, n_3, n_7

From the first analysis, we can observe that with the same replicas, our replica placement mechanism performs best, and has the smallest *TotalresponseTime*.

Second Analysis

As smaller replica degree means less cost of management, we compare the smallest *replicaDegree* of each mechanism to make sure that the response time of a single request is smaller than T_{upper}. The result is shown in Table 2.

Table 2. Results of second analysis

Mechanism	*replicaDegree*	Nodes to host replica
RTRM	3	n_2, n_5, n_6
Best Client	4	n_2, n_3, n_4, n_5
MinimizeExpectedUtil	3	n_1, n_2, n_5
MaximizeTimeDiffUtil	4	n_1, n_2, n_6, n_9
MinimizeMaxRisk	4	n_2, n_3, n_5, n_7

From the second analysis, we can see that our replica placement mechanism has the smallest *replciaDgree*. *MinimizeExpectedUtil* also has smallest *replicaDegree*, but its *TotalresponseTime* is bigger.

4.2 Simulation of Dynamic Replica Management Strategy

OptorSim is a scalable, configurable and programmable simulation tool for grid. It has five built-in replica management strategies. We compare our RTRM strategy with the five built-in replica strategies in OptorSim, and give the performance analysis. The simulation grid topology is shown in Fig. 2.

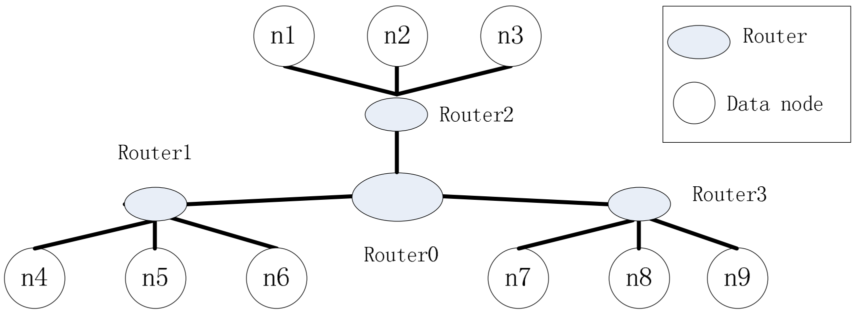

Fig. 2. The grid topology of simulation experiment

The simulation experiments are performed on a server machine, and the hardware and the software environment of the server machine is shown in Table 3.

Table 3. Environment of server machine

CPU	Quad-Core Intel Xeon 1.6GHz processors
Memory	4GB DDRII RAM
Hard Disk	320GB SATA II hard drive 7200RPM (ST3500418AS)
OS	64-bit CentOS 5.6 with Linux 2.6.18.8 kernel
OptorSim	OptorSim Release V 2.0.0

The simulation parameter configuration of the grid in our experiments is shown in Table 4.

Table 4. The configuration of simulation parameters

Parameters	value
Number of jobs	1000
Scheduler	File access cost + job queue access cost
optimizer	SimpleOptimiser LruOptimiser EcoModelOptimiserZipf DynamicOptimiser
Job delay	40000
Init file distribution	n_1, n_4, n_7
Max queue size	200

Fig. 3 shows the average job time of the six replica management strategies under three user access modes. In sequence mode, RTRM strategy is second best. In the random mode, RTRM strategy performs not so well. While in the Zipf distribution mode, our strategy performs best among all strategies.

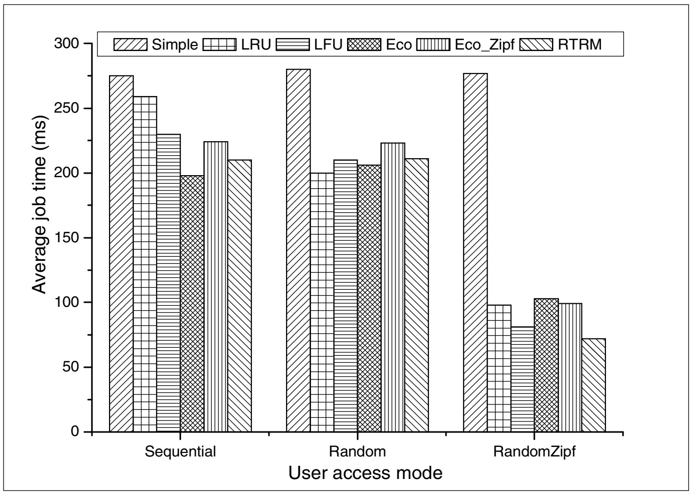

Fig. 3. Average job time

Fig. 4 shows the network utilization of the six replica management strategies under three user access modes. From the result, in any mode, RTRM strategy performs the best among the six strategies. This is because RTRM strategy takes the response time of a single request into consideration, making sure that the response time of any node smaller than T_{upper}.

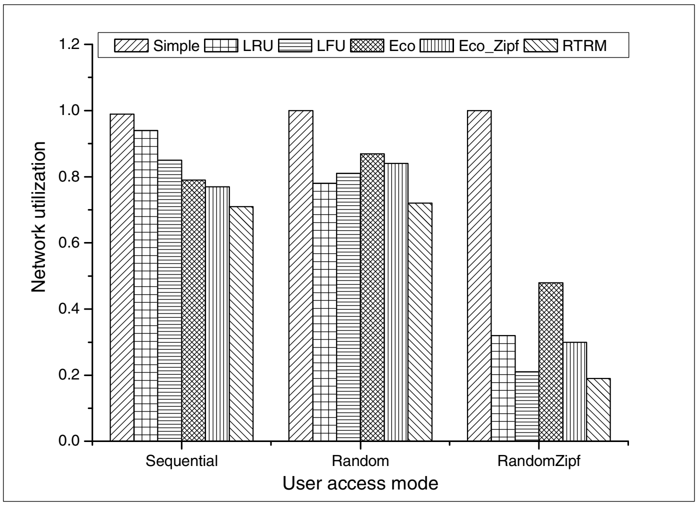

Fig. 4. Network utilization

Table 5 shows the number of total replicas of the six replica management strategies under three user access modes. Because the simple strategy has no replicas, the number of replicas of simple in Table 5 is always 0. From the table, we can see that the number of replica in RTRM strategy is far less than other five strategies in each access model. This is because we apply the reduction algorithm in the replica placement, and find the relatively better nodes to host the replicas for all the nodes in the system. Make sure the average service time is smaller than the threshold.

Table 5. Number of total replicas

	Sequential	Random	Random_Zipf
Simple	0	0	0
LRU	8851	6982	3583
LFU	6573	6751	3026
Eco	205	225	112
Eco_Zipf	425	512	374
RTRM	43	57	36

Through the analysis of simulation results, it can be deduced that RTRM strategy is very suitable for user access mode which follows Zipf distribution. The Zipf distribution means that user's access to file is coherent to time, which is very popular in the file sharing application of distributed storage system.

5 Conclusion and Future Work

Taking the response time of single request into consideration, we propose a response time-based replica management strategy referred to as RTRM, and it consists of replica creation method that can automatically increase the number of replicas based on the average response time. When a new request comes, RTRM will predict the bandwidth among the replica servers, and make the replica selection accordingly, and replica placement mechanism combing with the number of replicas and the network transfer time. In addition, we implement our dynamic replica management strategy in OptorSim. Through extensive simulations, we show that RTRM strategy behaves much better than the five built-in replica management strategies in OptorSim in terms of the network utilization and the service response time.

Finally, due to the limitation of OptorSim, the performance advantage of our replica selection method does not fully revealed in the simulation, but we believe that our replica selection method could achieve good performance and low response time, and provide rapid data download. In the future, we plan to apply our response time-based replica management strategy in HDFS [3], PVFS [11], pNFS [12], Gpfs [13], and LusterFS [14].

Acknowledgments. This paper is supported by National 863 High Technology Research and Development Program under grant No.2012AA010905 and National Science & Technology Pillar Program under grant No.2012BAH14F02.

References

1. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google File System. In: Proceedings of 19th ACM Symposium on Operating Systems Principles, pp. 29–43. ACM Press, New York (2003)
2. Sage, A.W., Scott, A.B., Ethan, L.M., Darrell, D.E.L., Carlos, M.: Ceph: A Scalable, High-Performance Distributed File System. In: Proceedings of 7th Conference on Operating System Design and Implementation (OSDI 2006), pp. 307–320. USENIX Press, Seattle (2006)
3. The Apache Software Foundation, Hadoop, `http://hadoop.apache.org/`
4. Sun, H., Wang, X., Zhou, B., Jia, Y., Wang, H., Zou, P.: The Storage Alliance Based Double-Layer Dynamic Replica Creation Strategy-SADDRES. Chinese Journal of Electronics 33(7), 1222–1226 (2003)
5. Hou, M.S., Wang, X.B., Lu, X.L.: A Novel Dynamic Replication Management Mechanism. Compute Science 33(9), 50–52 (2006)
6. Chang, R.S., Chang, H.P.: A Dynamic Data Replication Strategy Using Access-Weights in Data Grids. Journal of Supercomputing 45, 277–295 (2008)
7. Rahman, R.M., Barker, K., Alhajj, R.: Replica Placement in Data Grid: Considering Utility and Risk. In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), pp. 354–359. IEEE Press, Las Vegas (2005)
8. Ranganathan, K., Iamnitchi, A., Foster, I.: Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 376–381. IEEE/ACM, Berlin, Germany (2002)
9. Li, D., Xiao, N., Lu, X., Wang, Y., Lu, K.: Dynamic self-adaptive replica location method in data grids. Journal of Computer Research and Development 40(12), 1775–1780 (2003)
10. Bell, W.H., Cameron, D.G., Millar, A.P., Capozza, L., Stockinger, K., Zini, F.: OptorSim-A Grid Simulator for Studying Dynamic Data Replication Strategies. International Journal of High Performance Computing Applications 17(4), 403–416 (2003)
11. Ross, R.B., Rajeev, T.: Pvfs: A parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, pp. 391–430. USENIX Press, Atlanta (2000)
12. Hildebrand, D., Ward, L., Honeyman, P.: Large files, small writes, and pnfs. In: Proceedings of the 20th ACM International Conference on Supercomputing, pp. 116–124. ACM Press, New York (2006)
13. Schmuck, F., Haskin, R.: Gpfs: A shared-disk file system for large computing clusters. In: Proceedings of the First USENIX Conference on File and Storage Technologies, pp. 231–244. USENIX Press, Berkeley (2002)
14. Lustre: A scalable, High-performance File System, `http://www.lustre.ort/docs/lustre.pdf`