International Journal of Innovative Computing, Information and Control
Volume **13**, Number **3**, June **2017**

ICIC International ©2017 ISSN 1349-4198
pp. 1029–1036

RIPPLE LOAD BALANCING FOR IMPROVING LOAD IMBALANCE

Jui-Pin Yang

Department of Information Technology and Communication
Shih Chien University
No. 200, University Road, Neimen, Kaohsiung 84550, Taiwan
juipin.yang@g2.usc.edu.tw

Received November 2016; revised February 2017

Abstract. *Efficient load balancing is useful because load imbalance owing to simultaneous attempts to access the same data can significantly degrade the overall performance of storage systems. Existing algorithms need high communication overhead because they have to collect sufficient information such as load conditions in order to make correct decisions on dealing with load imbalance. In this study, we propose a novel load balancing scheme, which we refer to as ripple load balancing (RLB). RLB creates several replicas on distributed nodes to resolve load imbalance while keeping low communication overhead. When a node incurs a heavy load, a simple threshold-based scheduling algorithm is used to determine new destinations for dealing with the arriving requests. RLB triggers local balancing and finally achieves global balancing, i.e., a ripple effect. Simulation results show that RLB significantly improves load imbalance over a No Replica scheme and approaches to the performance of optimal load balancing under various load conditions.*
Keywords: Ripple load balancing, Threshold-based scheduling, Replica, Hotspot data

1. **Introduction.** Cloud storage services provide convenient data access so that users can read/write their data on demand. The main advantages of such services include low cost, quick response time, high data availability and so on. Consequently, cloud storage services play a key role in data storage [1]. In cloud computing, the main goal of load balancing is to uniformly distribute the loads of virtual machines across all computing nodes. Therefore, an efficient load balancing scheme can improve resource utilization and user satisfaction. Moreover, data can be processed efficiently with low delay [2]. Based on system information, dynamic load balancing schemes can distribute the loads of hotspot data to provide low resource response time and high utilization [3]. Thus, characteristics related to load balancing include distributing access requests among nodes, system performance, implementation complexity, response time, and resource utilization [4]. Typically, many users attempt to access popular data at the same time, thereby resulting in load imbalance. If an adequate load balancing scheme is applied to evenly distributing access requests to all nodes, it contributes to high system performance and low response time [5]. In addition to dynamic schemes, genetic algorithms can be employed to handle load imbalance [6]. In general, the genetic algorithms are useful to reduce migration and response times while maintaining simplicity of operation and power effect. However, applying genetic algorithms to storage systems with dynamic and varying load conditions is too difficult to practice. Existing schemes are known to have performance issues, high communication overhead, and long convergence times and are limited to specific load variations.

The control mechanisms of storage systems could be roughly classified as centralized control [7,8] and distributed control [9,10]. In centralized control, all requests arrive at a single entity responsible for determining which nodes will process the requests. In a

distributed control, all requests are handled by different nodes in the system and a scheduling algorithm determines the order in which requests are processed. Many well-known storage companies, such as IBM, offer load balancing services. For example, IBM published a study that explained how to use their SoftLayer load balancing service with the WordPress web application tier in the IBM cloud [11]. SoftLayer can be used to balance the loads of other web applications that are stateless and use a shared data store. The application state is stored in a shared relational database and a network-attached file system. The study also explained how to order and configure shared network-attached storage in the IBM cloud. As usual, data replication is used to improve load imbalance while keeping low data access latency in storage systems. Moreover, replica placement is one of the main topics. Different placement algorithms have different requirements on infrastructure such as storage capacity, and network bandwidth. A placement algorithm that meets the required performance goals with the lowest infrastructure cost was proposed [12].

In order to support the mutual consistency of replicated data, a model that considers the consistency control under failed communication links was proposed [13]. In this model, an algorithm transfers the constrained resource allocation into a mixed nonlinear integer program. However, the performance of the proposed algorithm will be deteriorated whenever the access pattern changes over time. DARE is a distributed adaptive data replication scheme which assigns data close to computation points as many as possible [14]. Accordingly, DARE assists the scheduler in achieving better data locality. DARE mainly evaluates the number of replicas for each file and determines the data placement based on probabilistic sampling and an aging algorithm. However, DARE may cause shortage of storage capacity. CDRM is used to capture the relationship between data availability and the number of replicas [15]. Therefore, it estimates the minimal number of replicas under a given data availability. Moreover, CDRM dynamically adjusts the number of replicas and their locations based on the variations of workload and node capacity. CDRM is cost-effective and outperforms default replication management of Hadoop distributed file system in terms of performance and load balancing. However, CDRM does not take the data de-replication and synchronization of writable data into account. In storage systems, internal network bandwidth is limited and hence the misplaced replicas may waste network bandwidth and accordingly deteriorate the overall performance of the storage systems. CRMS was proposed to resolve the above mentioned issue [16]. CRMS uses a model to capture the relationships between block access probability, replica location and network traffic. Next, the replica placement problem is formulated as a programming optimization problem. Finally, a heuristic algorithm is proposed to deal with the replica adjustments. In order to achieve excellent load balancing, SARM scheme was proposed [17]. This approach estimates the amount of hotspot data and then establishes dynamic number of replicas on adequate storage nodes. Due to the requirements of replication and estimation, the proposed scheme is suitable for high-performance storage systems.

In this study, we propose a static load balancing scheme for a general storage system because it uses a simple method to improve load imbalance while keeping reasonable load balancing. In this study, we propose a load balancing scheme for a storage system that uses a simple and efficient method to avoid congestion. The proposed load balancing scheme effectively avoids overloaded situations on a certain number of nodes. The remaining of this paper is organized as follows. The detailed design of RLB scheme is presented in Section 2. Simulation results are demonstrated and discussed in Section 3. Section 4 concludes the paper with future work.

2. **Ripple Load Balancing.** Figure 1 shows the architecture of a storage system comprising a metadata server (MDS) and N storage nodes [17]. MDS is in charge of dealing with arriving requests and storage nodes are in charge of data access. The arriving requests can be processed in two ways: centralized control or distributed control. For centralized control, all requests are handled by the MDS, which is responsible for determining which node should process the requests. To determine the best node, the MDS generally uses a scheduling algorithm to dispatch arriving requests according to specific requirements, such as loads and response time. Centralized control is considered a simple scheme, but its reliance on the MDS may result in a performance bottleneck. Moreover, centralized control architecture has a high communication overhead. However, for distributed control, each request is delivered to a default node that stores the required data. The default node that stores the original data is responsible for dispatching requests to other nodes with the same replicas. In general, hotspot data access patterns possess short-term and unpredictable characteristics. By duplicating replicas, a default node can adequately dispatch requests to other nodes, thereby improving load imbalance. The proposed ripple load balancing (RLB) scheme adopts distributed control for handling arriving requests that avoids the MDS server becoming a performance bottleneck in centralized control.

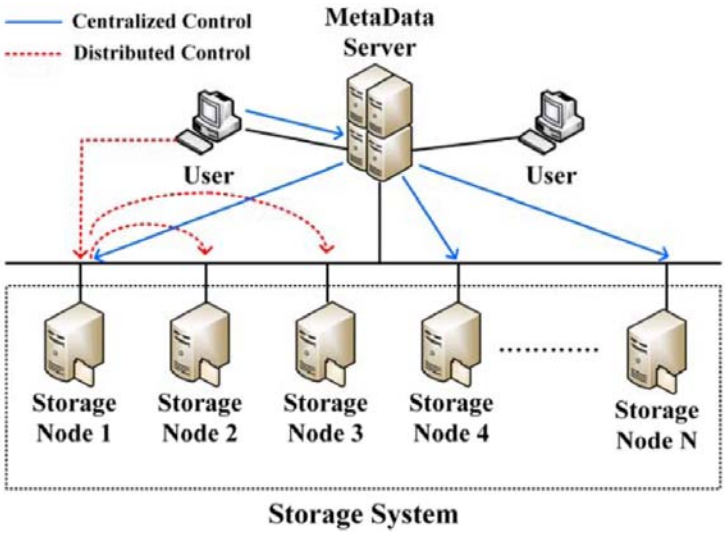

FIGURE 1. Storage system architecture

First, we estimated the average load of any node i whenever a request arrives or is processed using Equation (1). $Load_{i,new}$ denotes the updated estimate of the average load of node i comprising both hotspot and non-hotspot data. $Load_{i,old}$ is the load of node i prior to updating $Load_{i,new}$. $Load_{i,now}$ denotes the current load. Moreover, k_a is a parameter that prevents the estimates of $Load_{i,new}$ from having excessive dependency on short-term or long-term load conditions.

$$Load_{i,new} = k_a * Load_{i,old} + (1 - k_a) * Load_{i,now} \tag{1}$$

Next, we classified the average load of each node into corresponding thresholds according to Equation (2). T_i denotes the threshold of node i and $ceil()$ is a function that returns the smallest integer greater than or equal to a given number. C denotes the maximum processing capacity of simultaneous requests and k_b denotes a ratio of threshold. In other words, $k_b * C$ represents the range of the threshold. For example, when node 1 redirects a request to node 2, node 1 will label the threshold information in the request. Next, node 2 will compare its threshold to the threshold in the request when it receives the request.

If the threshold of node 2 is greater or equal to that of node 1, it will send a message with threshold information to node 1. By dividing the loads into different thresholds, RLB can avoid frequent updates of load variations. Therefore, RLB adopts an intelligent update method such that it creates low communication overhead.

$$T_i = ceil(Load_{i,new}/(k_b * C)) \tag{2}$$

When threshold-based scheduling is adopted, the node with the minimum threshold is selected via Equation (3). Then, the arriving request related to node i is dispatched to node p. Here, S_i denotes the set of nodes that store the replicas of node i

$$p = \{\min(T_i), i \in S_i\} \tag{3}$$

In general, hash algorithms can approximately distribute stored data evenly to each node. Thus, all nodes have approximately equivalent loads unless hotspot data requests occur. A new replica represents a certain number of requests to be redirected to another node. After completing the request scheduling, replica allocation is responsible for determining the locations of new replicas by selecting nodes according to Equation (4). Here, r_c denotes the number of replicas of node i. For simplicity, we assumed that all nodes have the same replicas. In addition, mod() is a function that obtains the modulus after division. If $\text{mod}(i + j * (N - 1)/r_c, N) = 0$, then the target node is node N. With additional replicas and efficient replica allocation, RLB can efficiently improve load imbalance.

$$S_i = \{\text{mod}(i + j * (N - 1)/r_c, N), \text{where } 1 \leq j \leq r_c\} \tag{4}$$

3. **Main Results.** We performed all simulations based on the storage system architecture, as shown in Figure 1. The request that generates the model of the hotspot data comprises an ON-OFF model. When a request arrives at the MDS, it will be transferred to the node with the required data. Similarly, we use the ON-OFF model to simulate the request variations for non-hotspot data in each node. In optimal load balancing (OLB), we assume that the MDS can obtain the real-time status of load conditions, and each node stores a replica of each data in the storage system. Therefore, the MDS always forwards requests to the node with the lowest load. Although OLB imposes a very large burden on the MDS and requires the maximum number of replicas, it demonstrates the best performance relative to handle load imbalance. In the following simulations, OLB served as a benchmark but it is too complicated to implement in current storage systems. Furthermore, we considered a scheme of no replicas for any data, i.e., a No Replica scheme.

Unless otherwise specified, we used the following settings in our simulations: $k_a = 0.8$, $N = 65$, $k_b = 0.01$, and $C = 25000$. In addition, we set the residing number of requests in each node to be between 7500 and 20000 in all experiments. When a request arrives at a node and the number of residing requests reaches maximum capacity, the new request is immediately discarded. Note that the simulation time in each experiment is of 20 hours. Other related parameter settings in each simulation are shown in corresponding figure. Related to generating requests from the hotspot data, on_off_pb denotes an ON-OFF parameter and off_on_pb denotes an OFF-ON parameter in the ON-OFF model. In addition, on_off_factor is a parameter used to adjust the load of generating requests. Related to background requests, decr_load_variation is a parameter used to control the request's decrement and incr_load_variation is a parameter used to control the request's increment.

In the first experiment, we considered the effect on load balancing when 8 nodes have high hotspot data loads. The experimental results are shown in Figure 2. In general, the excessive requests should be redirected to other nodes to distribute the load; otherwise,

large number of requests will simply be discarded. This is exactly the case in the No Replica scheme. As a result, nodes that did not contain hotspot data were under-utilized and nodes that contained hotspot data were over-utilized. Although the No Replica scheme requires only one replica, it demonstrated the worst load balancing, request loss, and long response time. OLB achieved the best load balancing because each node received an approximately equal amount of requests because it redirects the requests to the node with the lowest number of requests. As mentioned previously, OLB is too difficult to implement in an efficient storage system because it must maintain a replica of all data regardless of whether they are hotspot data. In addition, OLB leads to very low storage utilization, and each node must update its load conditions frequently.

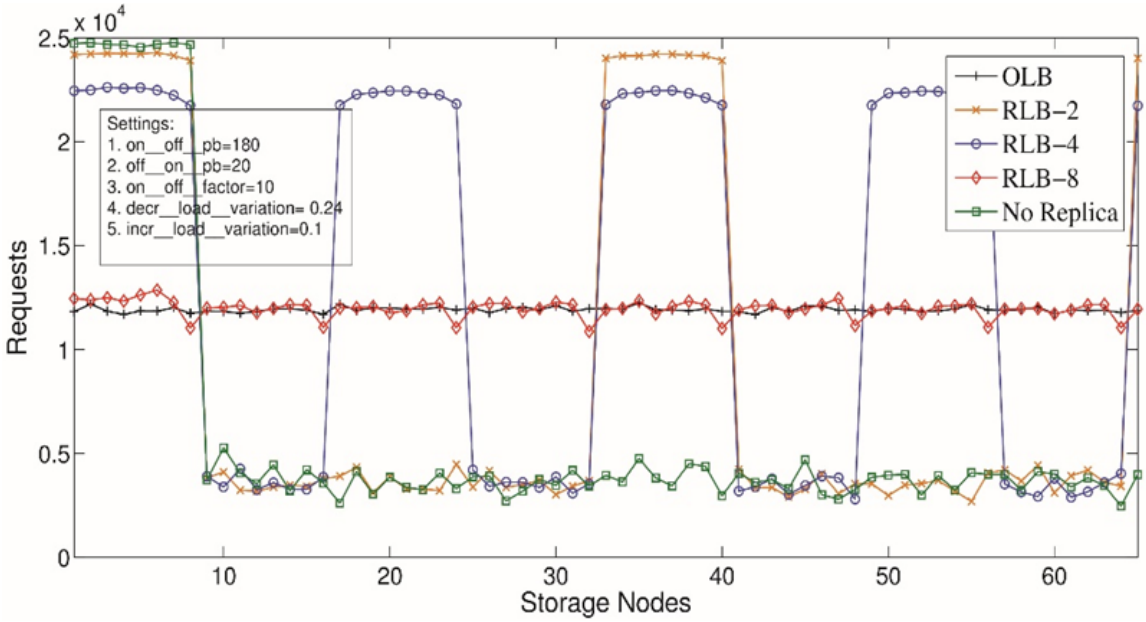

FIGURE 2. The average requests per node using eight nodes with high loads of hotspot data

With the RLB scheme, RLB-2 means that the hotspot data had two replicas (i.e., $r_c = 2$), whereas RLB-4 means that the hotspot data had four replicas, and so on. When the number of replicas was increased from two to eight, RLB demonstrated obviously improved load balancing because nodes with hotspot data can redirect requests to more candidates. Thus, it achieves better load balancing. In RLB-2, many requests arriving at nodes 1 to 8, 33 to 40, and 65 were discarded due to limited processing capability of those nodes. These were caused by high hotspot data loads at nodes 1 to 6 as a result of replica allocation. For example, node 1 redirected requests to nodes 33 and 65. In other words, the number of nodes was insufficient to relieve the load imbalance. RLB-4 has the same problem but it provides a little improvement. Finally, load balancing in RLB-8 was close to that of OLB. This means that the ripple effect was activated. Despite requiring fewer replicas, RLB tends to approach best load balancing and outperforms the No Replica scheme.

In the second experiment, we considered the effect on load balancing when 16 nodes have low hotspot data loads. The experimental results are shown in Figure 3. In RLB-2, many requests arriving at nodes 1 to 16, 33 to 48, and 65 were discarded due to limited processing capability of those nodes. These were caused by high hotspot data loads at nodes 1 to 16 as a result of replica allocation. For example, node 1 redirected requests to nodes 33 and 65. In other words, the number of nodes was insufficient to relieve the

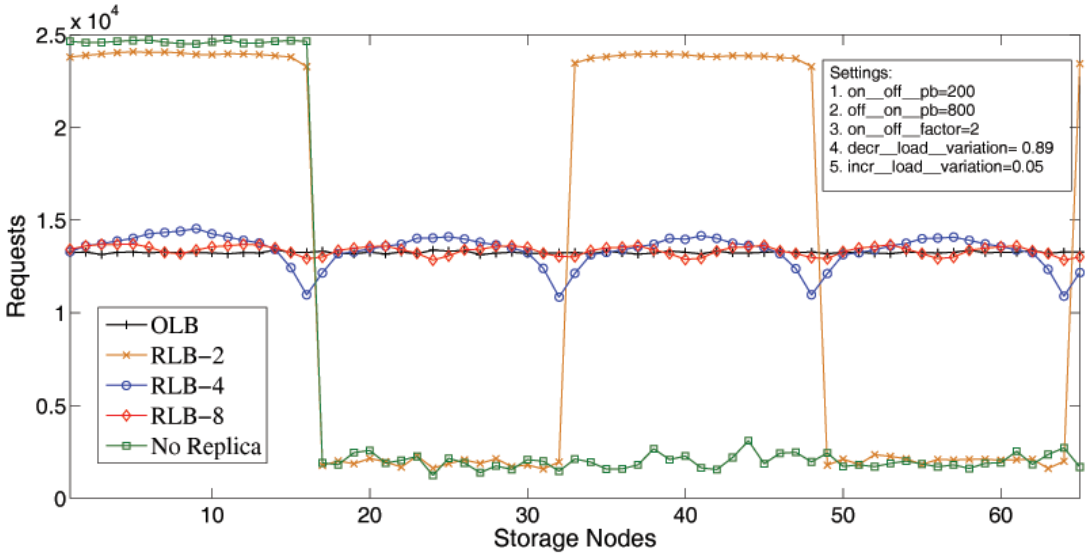

FIGURE 3. The average requests per node using sixteen nodes with low loads of hotspot data

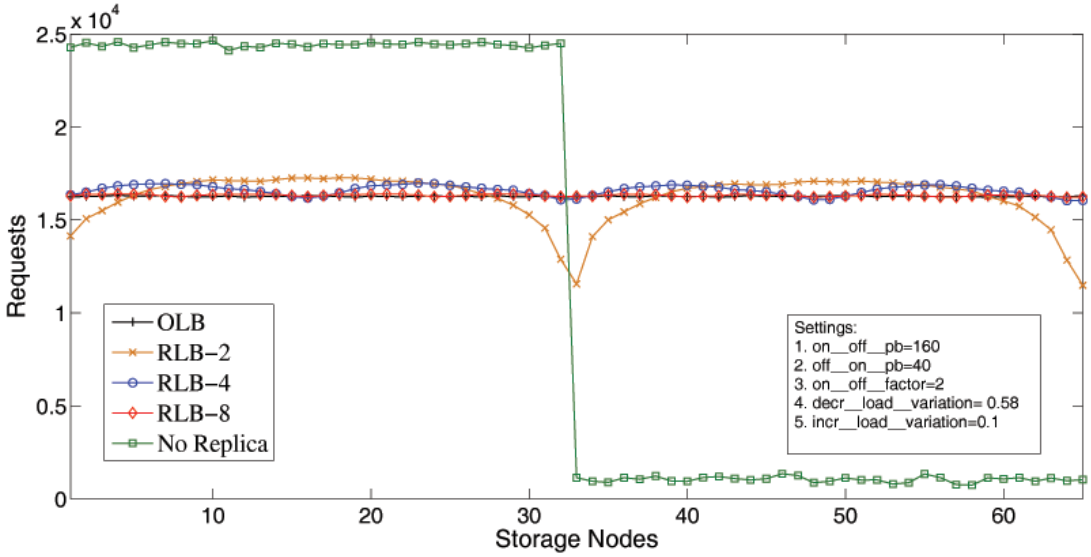

FIGURE 4. The average requests per node using thirty-two nodes with high loads of hotspot data

load imbalance. RLB-4 was much better than RLB-2 because the replicas were distributed fairly to all nodes. This means that the ripple effect was activated early as compared with the first experiment. Finally, load balancing in RLB-8 was close to that of OLB. However, the increased number of replicas showed little effect on improving load balancing. Despite requiring fewer replicas, RLB tends to approach best load balancing and outperforms the No Replica scheme.

In the last experiment, we considered the effects on load balancing when 32 nodes have high hotspot data loads. The experimental results are shown in Figure 4. Here, RLB-2, RLB-4, and RLB-8 all demonstrated better load balancing compared with the previous experiment because there are twice amount of nodes with hotspot data. In addition, the replicas were allocated fairly to nodes in the storage system. Therefore, the ripple effect worked much better. OLB maintained the best load balancing because it

always dispatches requests to a node with the lowest load. The No Replica scheme still demonstrated the worst performance. In other words, RLB shows better performance, especially for larger load variations. From Figure 2 and Figure 4, we conclude that RLB achieves excellent load balancing with a small number of replicas. In addition, RLB is simple to implement and consumes limited communication overhead under various load conditions. Thus, the proposed RLB scheme is suitable for deployment in general storage systems with reasonable performance and efficient storage usage.

4. **Conclusions.** Efficient load balancing is critical because it can avoid performance degradation by distributing load imbalance to appropriate nodes in a storage system. In this study, we proposed the RLB scheme. RLB works with limited communication overhead and requires a few replicas. The load sharing of RLB is much better than that of the No Replica scheme and approaches the performance of OLB with adequate number of replicas. However, OLB requires replicas of each data and hence it consumes a lot of storage space. In addition, real-time information updates are required by OLB. Accordingly, OLB is too complicated to implement on general storage systems. The RLB scheme satisfies the requirements of general storage systems owing to its simplicity and efficiency. In the future, we would like to consider user access behaviors, data types, and storage cost to develop an efficient and practical load balancing scheme for wide types of storage systems.

Acknowledgment. The author acknowledges the financial support from the Shih Chien University in Taiwan, under the grant numbers USC 105-08-01005.

REFERENCES

[1] K. Li, G. Xu, G. Zhao, Y. Dong and D. Wang, Cloud task scheduling based on load balancing ant colony optimization, *ChinaGrid*, pp.3-9, 2011.
[2] F. Xhafa, J. Carretero and A. Abraham, Genetic algorithm based schedulers for grid computing systems, *International Journal of Innovative Computing, Information and Control*, vol.3, no.5, pp.1053-1071, 2007.
[3] M. Katyal and A. Mishra, A comparative study of load balancing algorithms in cloud computing environment, *IJDCC*, vol.1, pp.5-14, 2013.
[4] R. G. Rajan and V. Jeyakrishnan, A survey on load balancing in cloud computing environments, *IJARCCE*, vol.2, pp.4726-4728, 2013.
[5] X. Xu, H. Yu and X. Cong, A QoS-constrained resource allocation game in federated cloud, *IMIS*, pp.268-275, 2013.
[6] A. Kanmani and R. Sukanesh, Adequate algorithm for effectual multi service load balancing in cloud based data storage, *JSIR*, vol.74, pp.614-617, 2015.
[7] M. Harchol-Balter, Task assignment with unknown duration, *J. ACM*, vol.49, pp.260-288, 2002.
[8] M. Mitzenmacher, The power of two choices in randomized load balancing, *IEEE Trans. Parallel Distrib. Syst.*, vol.12, pp.1094-1104, 2001.
[9] D. Breitgand, R. Cohen, A. Nahir and D. Raz, On cost-aware monitoring for self-adaptive load-sharing, *IEEE J. Sel. Areas Commun.*, vol.28, pp.70-83, 2010.
[10] S. Fischer, Distributed load balancing algorithm for adaptive channel allocation for cognitive radios, *CROWNCOM*, pp.508-513, 2007.
[11] J. M. Zhang, *Load Balancing Your Web Applications in SoftLayer*, IBM, 2014.
[12] M. Karlsson and C. Karamanolis, Choosing replica placement heuristics for wide-area systems, *Proc. of the 24th International Conference on Distributed Computing Systems*, pp.350-359, 2004.
[13] R. Tewari and N. Adam, Distributed file allocation with consistency constraints, *Proc. of the 12th International Conference on Distributed Computing Systems*, 1992.
[14] L. Abad, Y. Lu and R. Campbell, DARE: Adaptive data replication for efficient cluster scheduling, *Proc. of the IEEE International Conference on Cluster Computing*, pp.159-168, 2011.

[15] Q. Wei, B. Veeravalli, B. Gong, L. Zeng and D. Feng, CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster, *Proc. of the IEEE International Conference on Cluster Computing*, pp.188-196, 2010.

[16] K. Huang, D. Li and Y. Sun, CRMS: A centralized replication management scheme for cloud storage system, *2014 IEEE/CIC International Conference on Communications in China*, Shanghai, China, pp.344-348, 2014.

[17] J. P. Yang, Elastic load balancing using self-adaptive replication management, *IEEE Access*, no.99, p.1, 2016.