Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 4751829, 9 pages
http://dx.doi.org/10.1155/2016/4751829

Research Article

Efficient Load Balancing Using Active Replica Management in a Storage System

Jui-Pin Yang

Department of Information Technology and Communication, Shih-Chien University, 200 University Road, Neimen, Kaohsiung 84550, Taiwan

Correspondence should be addressed to Jui-Pin Yang; juipinyang@gmail.com

Received 22 March 2016; Revised 4 July 2016; Accepted 31 July 2016

Academic Editor: Pubudu N. Pathirana

Many algorithms can uniformly distribute data to storage nodes in a storage system. However, it cannot avoid load imbalance because data has different popularity. To resolve this issue, we propose a novel dynamic replication scheme, namely, Active Replica Management (ARM). ARM actively establishes optimal number of copies for hotspot data according to data access behaviors and then efficiently distributes copies to other storage nodes based on current amount of copies related to hotspot data. To improve storage utilization, ARM automatically and gradually dereplicates the useless copies of hotspot data when they become nonhotspot data. ARM resolves load imbalance by allocating dynamic copies to adequate storage nodes, and hence it can prevent partial storage nodes from overburdening. Simulation results demonstrate that ARM is an efficient scheme with excellent performance on load balancing, significantly closer to Optimal Load Balancing (OLB). In addition, ARM's performance outperforms both Static Load Balancing (SLB) and No Replica schemes.

1. Introduction

With rapid growth of various storage applications such as YouTube, Google Drive, and Megaupload, storage nodes that store popular data will become a performance bottleneck. Accordingly, load imbalance could lead to low resource utilization, high request loss rate, and long-latency response. In order to overcome such drawback, how to design an efficient load balancing scheme is a critical issue. A two-phase load balancing algorithm that combines opportunistic load balancing and load balance min-min scheduling algorithms achieves better executing efficiency in a system [1]. Unfortunately, this algorithm has a fatal defect because it could lead to system instability. A dynamic balancing algorithm evaluates the residual load rate and considers the current load condition of each server and processing capabilities as a result of different hardware configurations [2]. This algorithm has to maintain real-time state of each server, which not only increases communication overheads among different servers but also reduces deployment feasibility. Currently, replication-based schemes are the mainstream approach used to deal with load imbalance because they are also beneficial to enhance data reliability and decrease response time. A replica-aided load balancing scheme was proposed to enable the nodes in overlay networks, which can provide applications to users [3]. This scheme constructs a cost model to evaluate load balancing cost by considering the load, message number, and link latency at the same time. Furthermore, a performance tuning method is proposed to minimize cost while taking load balancing into account.

As for data reliability, if one node fails, users expect other alternative nodes to work instead [4, 5]. As a result, users can access their data all the time. In general, replication is a general way to satisfy such requirement. In this paper, we propose a simple and efficient load balancing scheme, namely, ARM. ARM can uniformly distribute the requests of hotspot data to other storage nodes by performing active replication and efficiently utilize storage resources by executing on-demand dereplication. By considering both short-term and long-term data access behaviors, ARM achieves excellent load balancing by establishing optimal number of copies for hotspot data on adequate storage nodes. Besides, ARM is beneficial for data reliability because of additional copies. The rest of this paper is organized as follows: Section 2 gives

a review of the related work. Section 3 presents the details of ARM scheme. We analyze the load balancing performance of the ARM and other compared schemes including OLB, SLB, and No Replica under different data access patterns in Section 4. Finally, Section 5 provides the concluding remarks and future work.

2. Related Work

Replication strategies have multipurpose efficiency on data reliability, response latency, and load balancing. Static replica strategy is easy to implement, which allocates first copy on a default node, and then randomly allocates other copies on some nodes [6]. This strategy improves data reliability while speeding up the data access operations. However, it does not consider available storage space whenever additional copies are needed. A block placement strategy is proposed to resolve above problem, which determines the replication nodes according to their real-time conditions [7]. This strategy achieves better load balancing but the allocated copies are permanently stored on the nodes which decreases the storage utilization. More importantly, the data access behaviors change frequently so that it cannot cope with dynamic environments. In addition, static strategies often ignore dynamic variations at run-time so that they do not have the ability to deal with load changes throughout run-time [8]. Central Load Balancing Decision Model (CLBDM) is a modification of Round Robin algorithm [9]. Unfortunately, it is unreliable because of unforeseen loops. Moreover, static strategies generally transfer only certain amount of data [10]. As a result, they cannot support sufficient fault tolerance [11].

Undoubtedly, a replication strategy that can dynamically allocate the copies is relatively useful to keep up with load changes in the system [12]. Six replication strategies were proposed including No Replica or Caching, Best Client, Cascading Replication, Plain Caching, Fast Spread, and Caching plus Cascading Replication. Only one original copy is available in No Replica or Caching no matter whether data belongs to hotspot data or not. Best Client creates one additional copy based on the number of requests. This strategy still cannot get rid of influences from hotspot data access because one copy is insufficient to improve load imbalance. Cascading Replication is similar to the Best Client. The main difference is that the new node for the copy is an ancestor of the Best Client. The mentioned strategies are infeasible to be deployed in current storage environments. Fast Spread strategy stores a copy on each node along its path to the client and hence the copy size is limited. A new optimization strategy utilizes a prediction function to evaluate the relative worth of files based on file access patterns [13]. It mainly creates and deletes a copy from a node with low storage space so as to avoid storage shortage. In another word, it mainly considers efficiency on storage management not load balancing. Dynamic strategies use some properties of the nodes inclusive of processing capabilities, load conditions, and network bandwidth. These strategies have to periodically check the node status and therefore they are too difficult to implement [14, 15]. In cloud computing environment, dynamic strategies can distribute work at run time and assign different weights to servers [16]. However, they are too complicated to achieve performance requirements [17]. Weight Least Connections (WLC) were proposed to assign tasks based on the number of connections for existing nodes [18]. In dynamic load balancing, if load balancer finds higher load of a node the consequent requests will be dispatched to the other nodes [19]. In order to efficiently control the load conditions, current status of nodes in the system should be maintained in time [20].

Two techniques were proposed to reduce the message traffic under overloading conditions [21]. The first technique replicates the messages at different topological regions of the network based on distance. The second technique demonstrates better performance by considering topology and physical proximities of the peers based on landmarks. However, both are infeasible to cloud environments because of network variations. A time-related replication algorithm was proposed, which achieves high data availability and less copy traffic by actively replicating the primary copy to reduce data loss probability [22]. This algorithm does not take creation and deletion of copies into account. A dynamic Minimum Access Cost (MAC) based replication strategy was proposed to improve dynamic minimum access cost by considering access frequency, network status, and average response time [23]. In order to accomplish optimal replication, MAC has to determine popular files and evaluate corresponding average response time. Next, MAC judges which files should be replicated. Finally, an appropriate node is selected to store files and hence it achieves low response time. MAC strategy is too difficult to be implemented because of inherently huge data and traffic. A load balancing mechanism based on ant colony and complex network theory in cloud computing federation is proposed to improve performance of current ant colony algorithms [24]. In addition, they qualitatively analyze the mechanism based on a prototype. A new content aware load balancing policy that consists of a hybrid approach of client aware policy and workload aware request distribution policy was proposed [25]. Moreover, a central load balancing policy for virtual machines was proposed to uniformly balance the load in a distributed environment [26]. In order to allocate data center resources to each virtual machine, statistic-based load balance scheme was proposed to balance the workload in the cloud [27]. This scheme makes use of the statistical prediction and resource evaluation and determines online storage resource allocations. As a result, it improves load balancing by predicting the resource demand of virtual machines not depending on Service Level Agreement (SLA) of each virtual machine.

In general, replication strategies could be classified based on their control policies, namely, centralized control and distributed control. Using centralized control, all jobs will arrive at a single control entity, which is in charge of determining which server is assigned to handle the job [28]. Using distributed control, jobs may arrive at any server in the system, which then needs to carry out specific scheduling policy to determine which server should be responsible for processing the job [29, 30]. Dynamic replication strategies that use distributed control have to collect information from

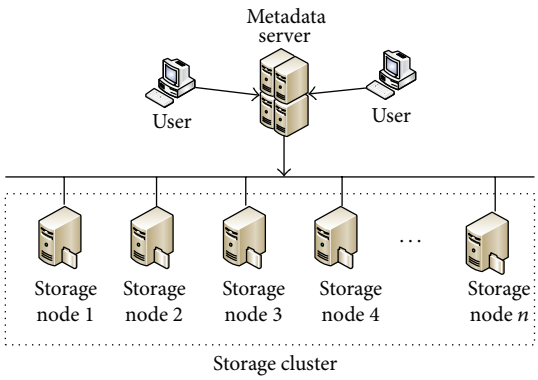

Figure 1: Storage system architecture.

the nodes in the distributed system and hence could lead to significant communication overheads and longer response time. A fully distributed implementation of the supermarket model performs well with appropriate tuning. In addition, it takes communication overheads into account. Centralized control is unsuitable for huge data processing centers because the single control entity will become the performance bottleneck, resulting in a single point of failure, longer response time, higher request loss rate, and so on. Therefore, ARM works with distributed control. Some strategies try to guess the required information when jobs are being processed, while other strategies require a priori knowledge so that they can effectively process jobs [31]. Although they do not require any such knowledge, they depend on other specific information such as the state of the system or the number of jobs awaiting in each server.

The work that we presented in this paper is an efficient scheme, namely, Active Replica Management (ARM). ARM utilizes active replica management to enhance load balancing while using distributed control. We verify the ARM's performance in a storage system. Particularly, two important features distinguish our approach from the other proposed approaches. First, ARM actively establishes optimal number of copies for hotspot data so that it can timely balance the load among storage nodes under time-varying data access patterns. Second, we further propose a dereplication method that can enhance storage utilization by eliminating unnecessary copies. In a word, our approach achieves approximately optimal load balancing by using dynamic copy assignment.

3. Active Replica Management

Figure 1 depicts a storage system architecture consisting of a metadata server and n storage nodes. When a user would like to access data, a query message is sent to the metadata server. Next, the metadata server will issue a response message to the user with the physical location of the required data. Finally, the user directly sends requests to the default storage nodes. In general, the access pattern on hotspot data often possesses short-term and unpredictable characteristics, and consequently it leads to load imbalance. According to the 80/20 rule, the load in the single metadata server is relatively higher than in the storage nodes. In ARM, the default storage nodes store original data and schedules arriving requests to other storage nodes with the same copies of data whenever hotspot data happens. Apparently, ARM avoids additional burden on the metadata server. In order to deal with excessive query messages from the users, several metadata servers can be deployed in the storage system. For instance, the quantity of metadata server may be proportional to the number of storage nodes. In this paper, a global namespace strategy was adopted to resolve the naming problem. All storage nodes within the storage system have an identical namespace. In another word, the path and name of each set of data on one storage node will be the same on every other storage nodes, regardless of where actual location of stored data is. When a user accesses desired data, the request is directed to the default storage node. Next, the storage node uses a request scheduling to determine which storage node is in charge of processing the request.

First, the definition of hotspot data is as follows. If the number of arriving requests for the same data exceeds H_{th} during a time interval T_d, then the data are identified as hotspot data. Otherwise, it is identified as nonhotspot data. H_{th} is denoted as a hotspot threshold. Furthermore, a copy of the hotspot data represents that an amount H_{th} of requests is assigned to another storage node. If $\text{count}_{i,j}^{T_c} > H_{\text{th}} * \text{Rep}_{i,j}^{\text{old}}$, the current number of copies related to the original data j in storage node i at time T_c is insufficient to cope with hotspot data access. The $\text{count}_{i,j}^{T_c}$ denotes the accumulating number of arriving requests related to data j in storage node i at time T_c. By monitoring the number of arriving requests, ARM can detect hotspot data in time. $\text{Rep}_{i,j}^{\text{old}}$ denotes the previous number of storage nodes having replicas related to data j in storage node i. $\text{Rep}_{i,j}^{\text{old}}$ is the value that exists before the updating of $\text{Rep}_{i,j}^{\text{new}}$. We use (1) to estimate the newly required number of copies while avoiding skew data access on storage node i, denoted by $\text{Rep}_{i,j}^{\text{incr}}$. The upper bound of $\text{Rep}_{i,j}^{\text{incr}}$ is limited by the number of total storage nodes in a storage system (SN_{sum}) minus the number of storage nodes with the copies:

$$\text{Rep}_{i,j}^{\text{incr}} = \min\left(\max\left(0, \text{ceil}\left(\frac{\text{count}_{i,j}^{mT_c} * \left(T_d/\left(m * T_c\right)\right) - H_{\text{th}} * \text{Rep}_{i,j}^{\text{old}}}{H_{\text{th}}}\right)\right), \text{SN}_{\text{sum}} - \text{Rep}_{i,j}^{\text{old}}\right). \quad (1)$$

We estimate the average loads of storage node i using (2). T_{slot} denotes the interarriving time of a request. The value $\text{avg}_i^{\text{new}}$ denotes the new average load of storage node i, and the value $\text{avg}_i^{\text{old}}$ is the value prior to the updating of

$\text{avg}_i^{\text{new}}$. Additionally, we assume that each request has the same processing time of T_{slot}. Furthermore, storage node k redirects the number of requests for hotspot data residing in storage node i, denoted by $RR_{k,i}$ during T_{slot}. If storage node j does not redirect any request to storage node i, then $RR_{j,i} = 0$. Finally, α is a parameter that adjusts the estimates of $\text{avg}_i^{\text{new}}$, and Request_i denotes the number of arriving requests in storage node i at the end of T_{slot}:

$$\text{avg}_i^{\text{new}} = \alpha * \left(\text{avg}_i^{\text{old}} + \text{Request}_i + \sum_{k=1\&k\neq i}^{\text{SN}_{\text{sum}}} RR_{k,i} - 1 \right) + (1-\alpha) * \text{avg}_i^{\text{old}}. \quad (2)$$

By simplifying (2), we obtain

$$\text{avg}_i^{\text{new}} = \text{avg}_i^{\text{old}} + \alpha * \left(\text{Request}_i + \sum_{k=1\&k\neq i}^{\text{SS}_{\text{sum}}} RR_{k,i} - 1 \right). \quad (3)$$

Next, we classify the average loads of storage nodes by using a particular partition method. This method should support sufficient granularity to determine which color the average loads should belong to. Assume that the maximum number of requests that a storage node can process is $RQ_{\max}$. The value γ_{green} denotes a ratio at which the load of the storage node constitutes underutilization, and γ_{red} denotes a ratio where the load of the storage node constitutes overutilization. Therefore, the control zone is in the region surrounding $[RQ_{\max} * \gamma_{\text{green}}, RQ_{\max} * \gamma_{\text{red}}]$. We divide the control zone into l_n levels, and each level is further divided into color layers. The number of color layers in each level depends on their scales. We denote the scale in level i by cs_i. When the loads of storage nodes are low, the scale should be larger for coarse granularity. Otherwise, the scale should be smaller for fine granularity. By comparing the color layers, the copy residing in the storage node with the highest color layer will be dereplicated first. The total number of color layers is described in (4), denoted by $\text{color}_{\text{sum}}$. Under underutilization and overutilization zone, they both have one level containing one color layer. In (4), two represents two color layers inclusive of underutilization and overutilization zone:

$$\text{color}_{\text{sum}} = 2 + \sum_{i=1}^{l_n} RQ_{\max} * \frac{\left(\gamma_{\text{red}} - \gamma_{\text{green}}\right)/l_n}{cs_i}. \quad (4)$$

To avoid excessive replication because of the burstiness of arriving requests, ARM constrains the allowable number of copies for each storage node according to their loads using (5). The value color_i denotes the color layer of storage node i, and Z denotes the set of copy limits in each level where $Z = \{Z_1, Z_2, Z_3, \ldots, Z_{l_n}\}$ and $Z_{k-1} < Z_k$. There are many ways to set the number of color layers in each level. The simplest approach is to make equal number of color layers in all levels. As we mentioned previously, the granularity can be very coarse for relatively low load. Based on simulations and analysis, we use nonlinear encoding in which lower levels are given smaller number of color layers (and thus finer granularity), while higher levels are given larger number of color layers. As a result, we use exponentially increasing model. Accordingly, $Z_k = p^{k-1}$ where p is a parameter used to allocate the number of color layers in each level:

$$\text{Rep}_{i,j}^{\text{incr}} = \begin{cases} 0 & \text{if } \text{color}_i = 1 \\ \min\left(\text{Rep}_{i,j}^{\text{incr}}, Z_{\lceil \text{color}_i/l_n \rceil}\right) & \text{if } 1 < \text{color}_i < \text{color}_{\text{sum}} \\ \text{Rep}_{i,j}^{\text{incr}} & \text{if } \text{color}_i = \text{color}_{\text{sum}}. \end{cases} \quad (5)$$

Next, the set of newly added storage nodes that possess the minimum number of copies related to all hotspot data are chosen according to (6). node_i represents storage node i, and $\text{SS}_{i,j}^{\text{cnt}}$ denotes the set of current storage nodes that already have a copy of data j in storage node i. Hot_i denotes the number of copies of total hotspot data in storage node i, and $\text{SS}_{i,j}^{\text{add}}$ denotes a set of newly added storage nodes that should be added to $\text{SS}_{i,j}^{\text{cnt}}$. In addition, these copies of hotspot data are identified as temporary items, whereas the original hotspot data are identified as permanent items. The difference is that permanent items are perpetually stored in storage nodes unless they are deleted by users, whereas temporary items might be removed when the original hotspot data became nonhotspot data. In a word, ARM distributes the copies based on the residing number of copies of all hotspot data in storage nodes. Traditionally, hash algorithms can approximately fairly distribute data to each storage node in a storage system, and hence all storage nodes have similar loads until hotspot data occur. ARM eliminates the load imbalance by allocating new copies according to the residing number of copies of all hotspot data in storage nodes. A new copy represents that a quota of arriving requests will be allotted to another storage node. Thus, ARM can efficiently achieve long-term load balancing. After the replication, ARM continuously monitors the elapsed time to determine whether more copies are needed:

$$\text{SS}_{i,j}^{\text{add}} = \sum_{i=1}^{\text{Rep}_{i,j}^{\text{incr}}} \left\{ \text{node}_i \mid \min\left(\text{Hot}_i\right) \text{ and } \text{node}_i \notin \text{SS}_{i,j}^{\text{cnt}} \right\}. \quad (6)$$

Finally, the number of copies (including temporary and permanent items of the hotspot data) related to data j in storage node i is updated using (7). In ARM, default storage node is in charge of dispatching the requests of hotspot data to other storage nodes with the same copy using a round robin fashion. In the end, ARM fairly distributes the requests among all storage nodes:

$$\text{Rep}_{i,j}^{\text{new}} = \text{Rep}_{i,j}^{\text{old}} + \text{Rep}_{i,j}^{\text{incr}}. \quad (7)$$

When the access frequencies of hotspot data decrease, ARM could invoke a dereplication method to enhance storage utilization. However, excessive dereplication will cause unnecessary resource waste. If $\text{count}_{i,j}^{b*T_d} \leq \min((\text{Rep}_{i,j}^{\text{old}} - f_d) * H_{\text{th}}, 0)$ is satisfied under each value of b, where $b = \{1, 2, 3, \ldots, k_b\}$, then (8) is used to choose a storage node

with the highest color layer, where k_b denotes the number of continuous time intervals T_d. Additionally, a dereplication factor f_d is used to achieve gradual removal of the copies of hotspot data. Therefore, $f_d \geq 1$:

$$SS_{i,j}^{\text{delete}} = \left\{ \text{node}_i \mid \max\left(\text{color}_i\right) \text{ and } \text{node}_i \in SS_{i,j}^{\text{cnt}} \right\}. \quad (8)$$

In addition to time changes, different hotspot data have various access frequencies, and therefore the dereplication method should not depend on the number of copies as the replication method does. Herein, ARM eliminates the imbalance access of requests by removing copies based on the current loads of storage nodes. With this, ARM efficiently achieves short-term load balancing. After a copy has been removed, default storage node will remove this record from its round robin list. ARM considers long-term and short-term data access behaviors so that it can provide stable and fair load sharing among storage nodes in a storage system. At the end, $\text{Rep}_{i,j}^{\text{new}}$ is updated by

$$\text{Rep}_{i,j}^{\text{new}} = \text{Rep}_{i,j}^{\text{old}} - 1. \quad (9)$$

4. Simulation Results

We perform all simulations based on the storage system architecture depicted in Figure 1. The generated user requests consist of an ON-OFF model. Similarly, we also use an ON-OFF model to simulate the request variations for nonhotspot data in each storage node. Related to the requests generating from the users, 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 requests. Related to the 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. The detailed explanations of request generating models are depicted in Algorithm 1.

In OLB, we assume the metadata server can obtain the real-time load conditions of all storage nodes, and each storage node in the storage system stores copies of all hotspot data. Therefore, the metadata server always forwards requests to the storage nodes with the lowest load. In another word, metadata server should be involved in request dispatching. SLB establishes fixed copies of hotspot data by selecting the storage nodes with minimum copies of hotspot data. Apparently, the SLB is a variant of the traditional static replica strategy. Besides, the metadata server should maintain a round robin list to redirect the requests to corresponding storage nodes. Thus, SLB could improve the load imbalance caused by hotspot data. Although OLB imposes a huge burden on the metadata server and requires the maximum number of copies, it demonstrates the best load balancing. In the following simulations, OLB serves as the benchmark of optimal load balancing. Furthermore, we also consider that hotspot data has no copy, namely, No Replica scheme.

Unless otherwise specified, we use the following parameter settings to all simulations. The ARM's parameters are set as follows: $T_d = 5$ minutes, $T_{\text{slot}} = 0.3$ seconds, $\text{SN}_{\text{sum}} = 128$,

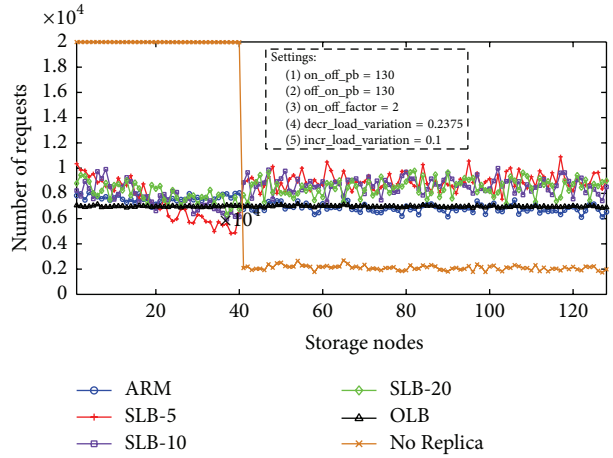

Figure 2: The number of requests versus storage nodes where forty storage nodes have hotspot data.

$H_{\text{th}} = 50$, $\alpha = 0.6$, $RQ_{\max} = 20000$, $\gamma_{\text{red}} = 0.80$, $\gamma_{\text{green}} = 0.16$, $l_n = 4$, $cs_i = 400 * p^{i-1}$, where $i = \{1, \ldots, l_n\}$, $p = 2$, $k_b = 6$, and $f_d = 3$. In addition, we set up the default number of requests in each storage node to be between 3200 and 16000 in random. When a request arrives at a storage node and the number of requests reaches the maximum capacity, the new request will be discarded. Also, the simulation time in the following experiments is of 20 hours. The other related parameter's settings in our simulations are attached to corresponding figures.

In the first experiment, 40 storage nodes have hotspot data and the simulation results are illustrated in Figure 2. Storage node 1 has the largest mean load of arriving requests (0.5) and storage node 40 has the lowest mean load of arriving requests (0.1923). In other words, the storage nodes with hotspot data are heavily congested ($0.5 + 0.1 > 0.2375$ and $0.1923 + 0.1 > 0.2375$) if no request is redirected to be processed by other storage nodes. In No Replica, the number of requests on the storage nodes with hotspot data reaches the maximum request capacity. However, those storage nodes are underutilized if no hotspot data resides. Although No Replica does not require any copies, it exhibits the worst load balancing. In addition, many requests are dropped which greatly degrades the service of quality. OLB demonstrates the best load balancing because each storage node has approximate number of requests. The reason is that it can timely redirect the requests to the storage nodes with the minimum number of requests because of full copies of hotspot data. However, OLB is difficult to be implemented. First, all storage nodes should keep a copy of data no matter whether they are hotspot data or not. In another word, OLB cannot recognize hotspot data. Second, OLB needs real-time updates on load status. SLB still cannot recognize the hotspot data so it wastes unnecessary storage space. SLB-5 means that each set of data has 5 copies (temporary copies), SLB-10 for 10 copies and so on. When the number of copies of hotspot data increases from 5 to 20, SLB achieves better load balancing. In SLB-5, storage nodes 1 to 20 have larger numbers of requests because the number of copies is insufficient to redirect

```
//pon_off(i): the probability that ON state changes to OFF state on hotspot data i
//poff_on(i): the probability that OFF state changes to ON state on hotspot data i
//on_off_pb: ON-OFF parameter
//off_on_pb: OFF-ON parameter
//on_off_factor: the parameter used to adjust the load of requests
//ON state: generates a new request to a storage node
//OFF state: no request is generated
pon_off(i) = l/(on_off_pb − on_off_factor ∗ (i − 1));
poff_on(i) = l/(off_on_pb + on_off_factor ∗ (i − 1));
//decr_load_variation: the parameter is used to control the request's decrement
//incr_load_variation: the parameter is used to control the request's increment
//ss_on_off_prob(j): the probability that ON state changes to OFF state in storage node j
//ss_off_on_prob(j): the probability that OFF state changes to ON state in storage node j
//burstiness_factor: the parameter used to control the burstiness of request changes
//ON state: increases a request inside a storage node
//OFF state: a request has been processed inside a storage node
for j = 1: storage_node
        request_change(j) = unifrnd(0, decr_load_variation + incr_load_variation);
        if(request_change(j) <= decr_load_variation)
           b = (1 − decr_load_variation)/2;
        else
           b = (1 − incr_load_variation)/2 + incr_load_variation;
        end
        ss_on_off_prob(j) = 1/ceil(b ∗ burstiness_factor);
        ss_off_on_prob(j) = 1/(burstiness_factor-ceil(b ∗ burstiness_factor));
end
```

Algorithm 1: Request generating models.

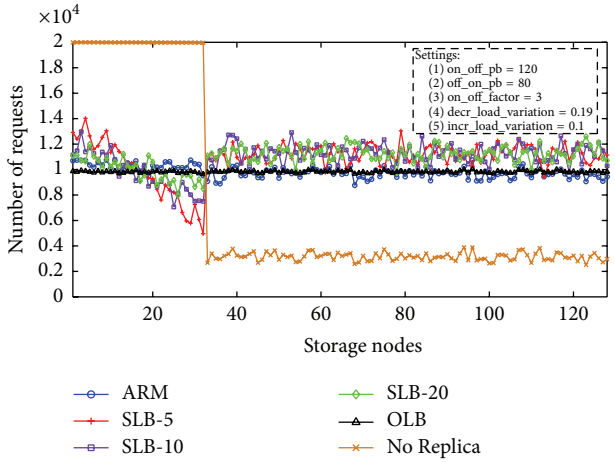

Figure 3: The number of requests versus storage nodes where thirty-two storage nodes have hotspot data under higher loads.

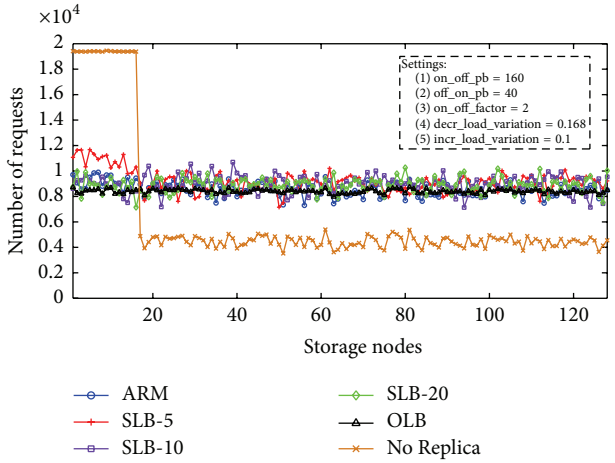

Figure 4: The number of requests versus storage nodes where sixteen storage nodes have hotspot data under different request conditions.

requests to other storage nodes. Inversely, storage nodes 21 to 40 redirect excessive requests to other storage nodes. As the number of copies increases, we find that the above disadvantage is improved. In ARM, the average number of copies of hotspot data is of 9.8585. However, this differs from the SLB because SLB-10 needs to replicate 10 copies of each set of data regardless of whether they belong to hotspot or nonhotspot data. Apparently, ARM requires relatively fewer number of copies but it tends to approach the load balancing of OLB and outperforms the SLB and No Replica schemes.

In the second experiment, we consider the effect on load balancing where 32 storage nodes have hotspot data with higher loads of arriving requests. The simulation results are illustrated in Figure 3. OLB repeatedly demonstrates the best load balancing and outperforms the other schemes. No Replica exhibits the worst performance and the nodes with hotspot data still encounter heavy congestion because the amount of arriving requests exceeds their processing capability. SLB-5 demonstrates worse load balancing as compared

with the first experiment because the variations of mean load of arriving requests is higher. For instance, the storage node 30 has the lowest number of requests. Apparently, storage nodes 1 to 16 have higher number of requests because their loads are higher than the average load. As compared with SLB-5, SLB-10 and SLB-20 both have relatively low effect because they both have more copies to resist the higher mean load. It is obvious that five copies (SLB-5 to SLB-10) have better improvement than ten copies (SLB-10 to SLB-20). Therefore, SLB cannot efficiently improve load balancing by increasing the number of copies related to hotspot data. ARM's performance is still close to OLB while demonstrating much better load balancing than SLB and No Replica schemes. In a word, ARM is able to achieve load balancing under higher loads.

In the third experiment, we consider the effect of dereplication method on load balancing where 16 storage nodes have hotspot data. In particular, no request is generated during time intervals [3, 5], [6, 8], [9, 11], [12, 14], and [15, 17] hours. The simulation results are illustrated in Figure 4. SLB-5, SLB-10, and SLB-20 all demonstrate better load balancing as compared with previous two experiments because the number of storage nodes with hotspot data is reduced. In SLB-5, the number of requests on storage nodes with hotspot data all exceeds that of OLB because their mean loads are larger than the overall mean load. OLB keeps the best load balancing because it always dispatches requests to the storage node with the lowest load. No Replica has better performance whose reason is the same as SLB. Repeatedly, ARM's performance approaches to OLB. In other words, ARM can efficiently replicate and dereplicate copies on demand while keeping excellent load balancing.

In the fourth experiment, we consider the effect of all storage nodes with higher load variations on load balancing where 24 storage nodes have hotspot data. The simulation results are illustrated in Figure 5. OLB demonstrates the best load balancing. In addition, such condition almost has no impact on load balancing. No Replica exhibits the worst load balancing. SLB improves load balancing along with the increment of copies but the improvement degrades quickly. In SLB-5, the number of requests on storage nodes with hotspot data randomly changes because of higher load variations. We find that ARM's performance still approaches to OLB. In other words, ARM adapts well to higher load variations.

In the fifth experiment, we consider the effect on fair load sharing when there are 64 storage nodes and all of them have hotspot data. In addition, the default number of requests in each storage node is between 6000 and 16000 in random. The simulation results are illustrated in Figure 6. In No Replica scheme, all storage nodes reach maximum capacity because it cannot dispatch the requests to other storage nodes and the default number of requests in each storage node is relatively higher than the previous experiments. Besides, many requests are dropped which greatly damages the performance of data access. Apparently, SLB improves load balancing along with the increasing number of copies because each storage node has hotspot data access. Accordingly, SLB-20 is better than SLB-10 and SLB-10 is better than SLB-5. In this experiment, ARM allocates 12.45 copies for each hotspot data which

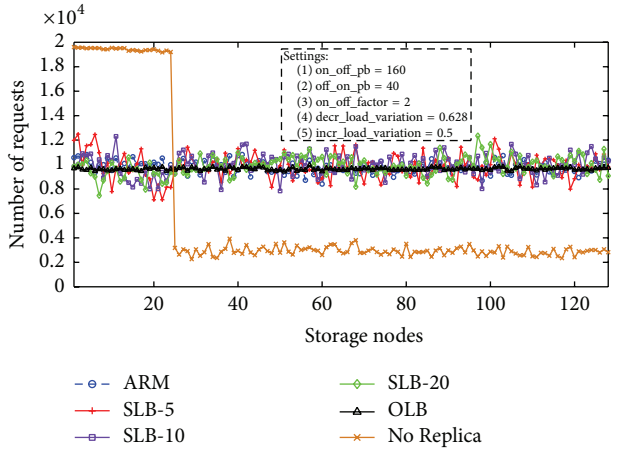

Figure 5: The number of requests versus storage nodes where twenty-four storage nodes have hotspot data under higher load variations.

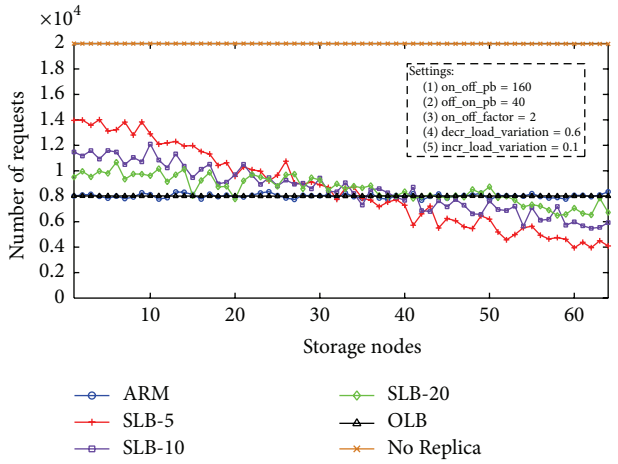

Figure 6: The number of requests versus storage nodes where all storage nodes have hotspot data.

is less than the SLB-20 with 20 copies. However, ARM achieves better load balancing than SLB-20. It means that ARM achieves better load balancing using smaller number of copies. With that, ARM has less communication overheads. More importantly, it is close to the OLB. From Figures 2–6, we conclude that ARM achieves excellent load balancing with adequate number of copies and less communication overheads under various conditions. As a result, ARM is suitable to be deployed in a storage system with performance requirements.

In the final experiment, we study how the load balancing is affected by different values of hotspot thresholds when storage nodes 1 to 32 have hotspot data. Storage node 1 has the largest average load of 800 requests per hour. The average load of these storage nodes decreases as the storage node number increases. Hence, storage node 32 has the smallest average load of 490 requests per hour. We illustrate the number of requests to storage nodes with hotspot data in Figure 7. When $H_{\text{th}} = 250$, those nodes have an average of 3.9913 copies.

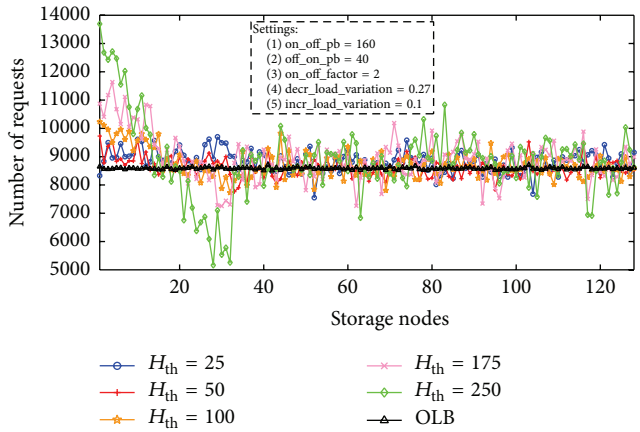

FIGURE 7: The number of requests versus storage nodes where thirty-two storage nodes have hotspot data under different hotspot thresholds.

In addition, each storage node has an approximate number of copies. There are two reasons for this phenomenon. First, storage node 32 holds the burstiness of generating requests, which results in a larger number of copies. Second, the continuous requests mean that no dereplication method is invoked. As a result, each copy of storage node 1 must handle approximately 200 requests, whereas storage node 32 needs to handle only approximately 120 requests. When H_{th} decreases, each hotspot data has more copies. When $H_{th} = 25$, $H_{th} = 50$, $H_{th} = 100$, or $H_{th} = 175$, the average number of copies is equal to 29.9650, 16.2359, 8.9849, or 5.7426, respectively. When the number of copies increases, the number of requests to each copy has an approximate value that enhances fair load sharing in general. However, $H_{th} = 50$ has better fair load sharing than $H_{th} = 25$ because too many copies can cause burstiness in redirecting requests to a storage node. From Figures 2–7, we conclude that ARM is applicable to various load conditions while preserving excellent load balancing accompanying with adequate copies of hotspot data.

5. Conclusions

In this paper, we propose a simple and efficient active replica management scheme, namely, ARM, and the goal is to leverage the loads among storage nodes in a storage system. This scheme actively establishes optimal number of copies for hotspot data according to data access conditions. Moreover, ARM dynamically adjusts unnecessary copies on demand. In a word, ARM not only improves load imbalance but also enhances storage utilization. Simulation results demonstrate that ARM can efficiently balance the loads among storage nodes under a variety of scenarios as compared with existing load balancing schemes inclusive of OLB, SLB, and No Replica. In future work, we would like to extend ARM by considering requests with different access priorities and by investigating the effect of load balancing while copies of hotspot data are stored across different storage systems. Finally, we will further study request loss rate and communication overheads.

Competing Interests

The author declares having no competing interests.

Acknowledgments

The author acknowledges the financial support from the Shih-Chien University and Ministry of Science and Technology in Taiwan, under the Grant nos. USC 104-05-05009 and MOST 103-2221-E-158-001, respectively.

References

[1] S.-C. Wang, K.-Q. Yan, W.-P. Liao, and S.-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *Proceedings of the 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT '10)*, pp. 108–113, Chengdu, China, July 2010.

[2] R. Tong and X. Zhu, "A load balancing strategy based on the combination of static and dynamic," in *Proceedings of the 2nd International Workshop on Database Technology and Applications (DBTA '10)*, pp. 1–4, Wuhan, China, November 2010.

[3] Y. Wang, Z. Zhou, L. Liu, and W. Wu, "Replica-aided load balancing in overlay networks," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 388–401, 2013.

[4] O. Olmez and A. Ramamoorthy, "Replication based storage systems with local repair," in *Proceedings of the 2013 International Symposium on Network Coding (NetCod '13)*, pp. 1–6, IEEE, Calgary, Canada, June 2013.

[5] S. Puntheeranurak and A. Chanpen, "SOA selection and replication based on QoS," in *Proceedings of the IEEE Region 10 Conference (TENCON '14)*, pp. 1–6, Bangkok, Thailand, October 2014.

[6] Apache Hadoop, 2015, http://hadoop.apche.org/.

[7] X. Ye, M. Huang, D. Zhu, and P. Xu, "A novel blocks placement strategy for hadoop," in *Proceedings of the IEEE/ACIS 11th International Conference on Computer and Information Science (ICIS '12)*, pp. 3–7, Shanghai, China, May 2012.

[8] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[9] N. Haryani and D. Jagli, "Dynamic method for load balancing in cloud computing," *IOSR Journal of Computer Engineering*, vol. 16, no. 4, pp. 23–28, 2014.

[10] N. J. Kansal and I. Chana, "Cloud load balancing techniques: a step towards green computing," *International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[11] M. T. Cse, "Comparison of load balancing algorithms in a Cloud Jaspreet kaur," *International Journal on Cloud Computing: Services and Architecture*, vol. 2, no. 3, pp. 1169–1173, 2012.

[12] I. Foster and K. Ranganathan, "Design and evaluation of dynamic replication strategies for high performance data grids," in *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, 2001.

[13] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, "Evaluating scheduling and replica optimisation strategies in OptorSim," in *Proceedings of the 4th International Workshop on Grid Computing (GRID '03)*, pp. 52–59, 2003.

[14] V. Sakthivelmurugan, A. Saraswathi, and R. Shahana, "Enhanced load balancing technique in public cloud," *International Journal of Research in Engineering & Advanced Technology*, vol. 2, no. 2, pp. 1–4, 2014.

[15] R. Somani and J. Ojha, "A hybrid approach for VM load balancing in cloud using cloudsim," *International Journal of Science, Engineering and Technology Research*, vol. 3, no. 6, pp. 1734–1739, 2014.

[16] R. Tong and X. Zhu, "A load balancing strategy based on the combination of static and dynamic," in *Proceedings of the 2nd International Workshop on Database Technology and Applications (DBTA '10)*, pp. 1–4, IEEE, Wuhan, China, November 2010.

[17] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[18] Y. A. Younis, M. Merabti, and K. Kifayat, "Secure cloud computing for critical infrastructure: a survey," Tech. Rep., Liverpool John Moores University, Liverpool, UK, 2013.

[19] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 551–556, Perth, Australia, April 2010.

[20] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and privacy in cloud computing: a survey," in *Proceedings of the 6th International Conference on Semantics, Knowledge and Grid (SKG '10)*, pp. 105–112, Beijing, China, November 2010.

[21] H.-C. Hsiao, H.-Y. Chung, H. Shen, and Y.-C. Chao, "Load rebalancing for distributed file systems in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 5, pp. 951–962, 2013.

[22] K. Kim, "Time-related replication for p2p storage system," in *Proceedings of the 7th International Conference on Networking (ICN '08)*, pp. 351–356, Cancun, Mexico, April 2008.

[23] X. Sun, J. Zheng, Q. Liu, and Y. Liu, "Dynamic data replication based on access cost in distributed systems," in *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology (ICCIT '09)*, pp. 829–834, IEEE, Seoul, Republic of Korea, November 2009.

[24] Z. Zhang and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation," in *Proceedings of the 2010 2nd International Conference on Industrial Mechatronics and Automation (ICIMA '10)*, pp. 240–243, IEEE, Wuhan, China, May 2010.

[25] H. Mehta, P. Kanungo, and M. Chandwani, "Decentralized content aware load balancing algorithm for distributed computing environments," in *Proceedings of the International Conference and Workshop on Emerging Trends in Technology (ICWET '11)*, pp. 370–375, February 2011.

[26] A. Bhadani and S. Chaudhary, "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud," in *Proceedings of the 3rd Annual ACM Bangalore Conference (COMPUTE '10)*, ACM, Bangalore, India, January 2010.

[27] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, "A statistical based resource allocation scheme in cloud," in *Proceedings of the 2011 International Conference on Cloud and Service Computing (CSC '11)*, pp. 266–273, IEEE, Hong Kong, December 2011.

[28] M. Harchol-Balter, "Task assignment with unknown duration," *Journal of the ACM*, vol. 49, no. 2, pp. 260–288, 2002.

[29] D. Breitgand, R. Cohen, A. Nahir, and D. Raz, "On cost-aware monitoring for self-adaptive load sharing," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 1, pp. 70–83, 2010.

[30] S. Fischer, M. Petrova, P. Mähönen, and B. Vöcking, "Distributed load balancing algorithm for adaptive channel allocation for cognitive radios," in *Proceedings of the 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom '07)*, pp. 508–513, Orlando, Fla, USA, August 2007.

[31] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 864–879, 2006.