WABRM: A Work-Load Aware Balancing and Resource Management Framework for Swift on Cloud

Zhenhua Wang[1], Haopeng Chen[1], and Yunmeng Ban[2]

[1] Shanghai Jiao Tong University, Shanghai, China
[2] University of Massachusetts at Amherst, Amherst, USA
aspiration@foxmail.com, chen-hp@sjtu.edu.cn,
banyunmeng@gmail.com

Abstract. Fueled by increasing demand of big data processing, distributed storage systems have been more and more widely used by enterprises. However, in these systems, few storage nodes holding enormous amount of hotspot data could become bottlenecks. This stems from the fact that most typical distributed storage systems mainly provide data amount balancing mechanisms without considering the difference of access load between different storage nodes. To eliminate bottlenecks and tune the performance, there is a demand for such systems to employ a work-load aware balancing and resource management framework to optimize the performance and computation resource utilization.

In this paper, we propose WABRM, a load balancing and resource management framework for *W*ork-load *A*ware *B*alancing and *R*esource *M*anagement in Swift, a typical distributed storage system. By designing such an optimization framework, it is possible to eliminate bottlenecks caused by hotspot data. Our experimental results show that the framework can achieve its goals.

Keywords: distributed storage system, Swift, work-load balancing, resource management.

1 Introduction

The distributed storage system significantly improves the capacity of big data storage, process and security. Swift [1], as a well-known and typical distributed storage system, is playing an important role in cloud storage. In Swift, there are mainly two kinds of nodes, including proxies and storage nodes. Data requests are sent to proxies and proxies fetch data stored in storage nodes to respond to users.

Concurrently, virtualization technology is making a significant impact on how resource are used and managed in a cloud computing platform. Several virtualization solutions (Xen [2], XenServer [3] and VirtualBox) are getting more and more mature in resource management.

Load balancing mechanisms for distributed system are also very important. There are mainly three default load balancing mechanisms in Swift. Firstly, scalable proxy mechanism allows users to set up more than one proxy to distribute requests from users to these proxies. Secondly, replica load balancing mechanism balances the work-load through responding with data replicas stored in different storage nodes.

J. Kołodziej et al. (Eds.): ICA3PP 2013, Part I, LNCS 8285, pp. 446–457, 2013.

Finally, data amount balancing mechanism tries to distribute data to all the storage nodes evenly. If the access load of each data is nearly the same, the work-load is balanced. In addition, there are also plenty of relative researches, such as research work [4, 5]. Their balancing targets are similar to the mechanisms' of Swift. However, almost all the previous researches remain the static data storage mechanism unmodified. And how to achieve the goal of load balancing dynamically in distributed storage systems has not been well studied. Hence we propose the framework named WABRM.

WABRM mainly contains three aspects, including discovery of work-load exception, algorithms of workload balancing and data migration method. Some relative work introductions are as follows.

Discovery of work-load exception is the basis of work-load aware balancing. Compare with the monitoring architecture of [6], in WABRM, each node monitors itself. Consequently, it is much easier to locate the hotspot data. In [7], for each chunk in MongoDB [8], its access-load is evaluated by the numbers of various operations on it. In WABRM, the access-load is evaluated by computation resource utilization, which can more objectively reflect its actual access-load. In [9], the exception of work-load is detected based on the predicted access load. It is good, but large amount of historical records is needed to guarantee its precision.

Algorithms of work-load balancing are the cores of work-load aware balancing framework. In [10], the files are divided into several zones according to the foreseen work-load in order to balance the access load. However, it is static since the location of a file will not be changed once it is stored into the zone. In [11], the data are dynamically re-partitioned to facilitate rapid data balancing by a graph theoretic way. Unfortunately, it is time consuming in some situations. In WABRM, we propose novel work-load balancing algorithms, which is dynamic and efficient.

Data migration is one way to achieve the goal of dynamic work-load balancing. In [12], a cost aware method is designed to minimize the interference between virtual machines. But the amount of data to be migrated is not reduced. In [13] and [14], a location-aware method is proposed to save energy when performing data migration in large-scale datacenters. Actually, we aim to balance access-load instead of storage amount. Thus, we can achieve this goal through virtual machine migration. WABRM adopts virtual machine live migration as its migration method.

In summary, work-load aware balancing is crucial for storage applications with hotspot data.

The contributions of this paper are summarized as follows. Firstly, we propose a work-load aware balancing and resource management framework based on the virtualization technology, which can be applied to Swift. WABRM is lightweight and requires no source code change in the guest OS and storage application. Secondly, we implement dynamic work-load balancing mechanisms for physical machines and virtual machines in WABRM. Finally, we conduct an experiment to demonstrate the effectiveness of WABRM in tuning the performance of Swift when hotspot data exist.

The rest of the paper is organized as follows. Section 2 describes motivating experiments to show the poor performance of the default load balancing mechanisms of Swift when hotspot data exist. Section 3 introduces the framework of WABRM as well as the design and implementation of the algorithms we integrated it. Section 4 presents the experimental results and analysis. Section 5 draws some conclusions.

2 Motivation

This section mainly describes the motivating experiment to show the poor work-load balancing performance of Swift when hotspot data exist as well as the analysis of the problem. We conduct the experiment on 15 virtual machines created by XenServer. They are represented by VM1, VM2…VM15 respectively. Storage nodes of Swift are deployed in these virtual machines. This experiment is simple but effective.

In this experiment, the number of replicas of a file is set to be 3, which is mostly accepted by the industry. Firstly, we upload some files to Swift. And we observe File A and its replicas are stored in VM1, VM9 and VM13 while File B and its replicas are stored in VM3, VM9 and VM14. We notice that VM9 stores both replicas of File A and File B. Through simulation of requests for File A and File B, we can observer the work-load difference of nodes. To simulate data access, Pylot [15], a web stress test tool is used to simulate clients. In this experiment, 100 clients and another 100 clients are simulated to fetch File A and File B respectively. The simulation of client requests is last for 15 minutes. During the 15 minutes, the concurrent 200 simulated users send their requests to Swift continuously and the interval between two requests is 100ms.

As Figure 1 shown, the work-load of VM1 and VM2 is significantly different and VM1 is much higher than VM2. The reasons are as follows. VM1 stores a replica of File A while VM2 stores no replicas of File A and File B, consequently, VM1 need more computation resource to respond to user requests and its work-load is much heavier. Even though there are three balancing mechanisms in Swift, they don't work in this scenario.

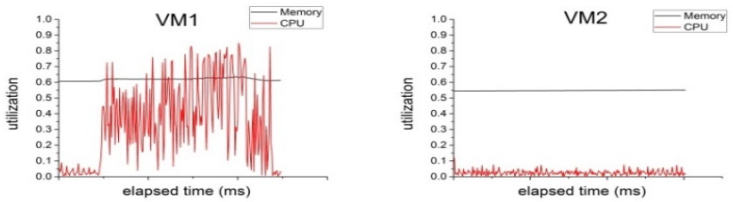

Fig. 1. Comparison of busy and free storage nodes

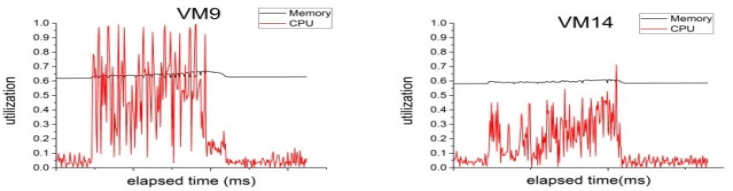

Fig. 2. Comparison of busy storage nodes

As Figure 2 shown, both VM9 and VM14 are busy, however, their work-load is obviously different. As mentioned before, VM9 stores replicas of File A and File B and VM14 only stores replica of File B, as a result, work-load of VM9 is obviously higher than VM14's. In Swift, this work-load imbalance is caused by its imperfect replica load balancing mechanism. Ideally, the storage system should first choose the replica in the storage node with the lightest work-load to respond. Actually, this imbalance can be resolved through rescheduling the response replica. To reduce the work-load, another solution is dynamic computation allocation, which is used in WABRM.

As Figure 1 and Figure 2 shown, the work-load of different storage nodes is different. So it is irrational to distribute computation resource to each storage node evenly. In WABRM, with Swift storage node deployed in virtual machines, we can allocate the resource to the storage nodes elastically. Further, a physical machine's resource utilization can be optimized through Split and Merge algorithms, which will be discussed in this paper.

Based on the aforementioned analysis, we try to optimize the work-load balancing of storage node in Swift through virtualization technology, a novel method. Through our method, the system performance is tuned and computation resource utilization is improved.

3 WABRM Architecture

This section mainly discusses the design of WABRM as well as the algorithms we have incorporated it. Before introducing WABRM architecture, we first present the work-load balancing model of WABRM.

3.1 Work-Load Balancing Model of WABRM

In traditional distributed storage systems, there is only one mapping of data to storage locations. Data are distributed and stored into different storage nodes according to some mapping rules, e.g. hash values. The mapping rule is static and difficult to change. To achieve the goal of dynamic work-load balancing, in WABRM, in addition to such mapping, there is a mapping of storage nodes to physical machines, which is dynamic and easy to modify. In this model, storage nodes are deployed in virtual machines and virtual machines reside in physical machines. By this way, data are divided into several much smaller subset, which improves the efficiency of locating hotspot data. Through dynamic changing the mapping of virtual machines to physical machines, work-load is balanced in physical machines.

As Figure 3 shown, WABRM is mainly composed of two layers, including physical layer and virtual layer. In this paper, a virtual machine with WABRM and Swift deployed is called a virtual node and a physical machine with WABRM and virtualization server deployed is called a physical node.

In virtual layer, WABRM is responsible for monitoring the work-load of a virtual node and scheduling the computation resource allocated to it according to its work-load

through interacting with the WABRM in the physical node. And in physical layer, WABRM is responsible for monitoring the work-load of a physical node and regulating it through interacting with other physical nodes and virtual node migration.

To achieve the optimization goal, we implement algorithms for virtual layer and physical layer.

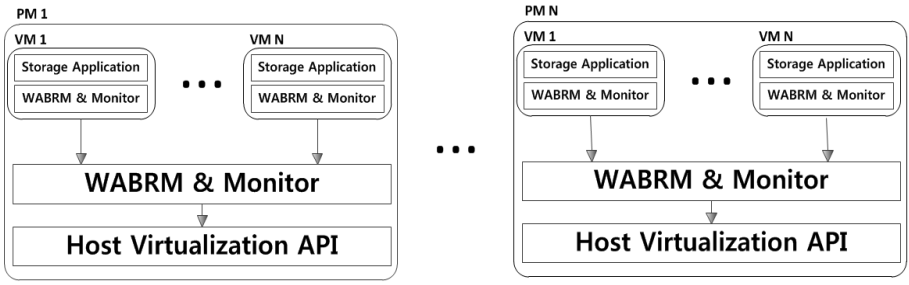

Fig. 3. WABRM Architecture

3.2 Work-Load Monitoring and Analysis

To achieve the goal of work-load aware balancing and resource management, work-load monitor is necessary in the framework. It provides a foundation for the algorithms we proposed. Through the monitor, work-load states of physical nodes and virtual nodes are collected and saved for further work-load analysis. In WABRM, there are two kinds of work-load monitors for different monitoring objectives. Work-load monitor for virtual node intermittently collects computation resource utilization information, including CPU and Memory utilization for the input of the algorithms for virtual layer. And Work-load monitor for physical node intermittently collects NetworkIO utilization in addition to the computation resource utilization information collected by work-load monitor for virtual node for the input of the algorithms for physical layer.

The goal of work-load analysis is to learn the work-load state of a node. To simplify the problem, in WABRM, we define three work-load types of a node, including underloaded, normal and overloaded. Generally, a node's work-load state can be represented by a computation resource utilization vector, which consists of CPU utilization, memory utilization, etc.

The current state vector of the node is calculated by historical monitoring states collected by work-load monitor of the node, since it is unreasonable to determine a node's state through single monitoring. Currently, in WABRM, computation resource utilizations in the state vector are the averages of their recent historical monitoring values. However, WABRM provides an interface for the realization of this node state calculation algorithm, any new algorithm, e.g. algorithms based on prediction can be integrated in to WABRM.

If the state vector is calculated, we use the following function to determine the load type of the node.

$$L(\mathrm{n}) = \begin{cases} overloaded & \exists \mathrm{i}, \mathrm{u}_i > \mathrm{opt}_i \\ underloaded & \sum_{i=1}^{n} \mu_i(\mathrm{opt}_i - \mathrm{u}_i) > t \\ normal & otherwise \end{cases} \quad (1)$$

In this function, u_i represents the utilization of Resource i, μ_i represents the weight of Resource i, opt_i represents the defined ideal utilization of Resource i and t represents the threshold of underloaded. Since excessive utilization of any resource can lead to poor system performance, the rule for determining overloaded is rational. And the rule for determining underloaded takes all kinds of resource utilization into consideration as well as provides weight for elastic configuration. Therefore, it is rational, too.

Apparently, excessive resource utilization can cause poor system performance. Based on this principle, WABRM optimizes the performance through regulation of the computation resource utilization. However, low resource utilization doesn't mean high performance, since critical resources may lead to low resource utilization. This problem is caused by the design of the native system. So WABRM may not improve the native system performance by this way, but it can tune the performance.

3.3 Algorithms

Currently, WABRM mainly integrates algorithms for physical layer and virtual layer. Other algorithms are applicable to WABRM architecture as well.

Resource Reallocate Algorithm

To regulate a virtual node, Resource Reallocate Algorithm (RRA) is invoked in the virtual node when it is determined to be overloaded or underloaded.

Because of the work-load difference between different storage nodes, it is irrational to distribute computation resource to each virtual node evenly. RRA reallocates the computation resource based on the load type of a virtual node. Generally, an overloaded node should be allocated more computation resource while an underloaded node's computation resource should be deallocated. Thus, RRA is designed to achieve this goal in virtual nodes.

To calculate the reallocation resource, we use the following function:

$$\mathrm{r}_i = \frac{\mathrm{C}_i * \mathrm{u}_i}{\mathrm{opt}_i} \quad (2)$$

In this function, C_i represents the capacity of original allocated Resource i, u_i represents the current utilization of Resource i and opt_i represents the ideal utilization of Resource i. Theoretically, the utilization of resource will reach ideal value after operation and the load type of the node will become normal. Thus, busy nodes will be allocated more computation resource while free nodes' resource will be deallocated. After calculation, the operation sends the request to the physical node in which the virtual node resides and the request is processed by the physical node.

If a physical node doesn't have enough computation resource to reallocate to an overloaded virtual node, it will allocate its resource to the virtual node as much as possible. Although it is possible that the virtual node remains overloaded after regulating, the physical node and virtual node will be further regulated by the follows algorithms for physical layer.

Split and Merge Algorithms

To regulate a physical node, Split Algorithm (SA) is invoked in the physical node when it is determined to be overloaded while Merge Algorithm (MA) is invoked when it is determined to be underloaded.

An overloaded physical node may be caused by the following reasons. 1. Numerous overloaded virtual nodes reside in it. 2. Some extremely overloaded virtual nodes reside in it. Thus, to regulate an overloaded physical node, it is rational to regulate the virtual nodes in it. SA is designed to achieve this goal. SA relieves an overloaded physical node's work-load through virtual node migration.

Before virtual node migration, SA decides the virtual node to be migrated. The policy meets two constraints: the number of virtual node to be removed is as little as possible and after migration, the physical node is not overloaded. To meet these constraints, SA figures out the nodes through backtracking. When the virtual nodes are selected, Pair Algorithm is invoked to pair another physical node as target for migration.

Correspondently, when a physical node is determined to be underloaded, MA is invoked and tries to move all the virtual nodes in it to other physical nodes. However, MA is not executed immediately since there may be some overloaded physical nodes searching for underloaded nodes for SA. Therefore, it waits for a specified period and if there are no requests from overloaded nodes, MA continues and invokes PA to determine the virtual node migration program. If the migration is successful, the physical node is empty. Hence, it can go to sleep for energy saving.

Pair Algorithm

To regulate a physical node, Pair Algorithm (PA) is invoked in the physical node when it is determined to be overloaded or underloaded.

If the work-load exceptions of some physical nodes in the cluster are detected, we try to dispose of them through interaction with other physical nodes in the global cluster. PA is designed to achieve this goal.

To an overloaded physical node, PA tries to pair a physical node or boot up a new physical node for it. Through migration of some virtual nodes in it to the paired physical node, its work-load is reduced.

To an underloaded physical node, PA tries to pair one or more physical nodes for it. Through migration of all the virtual nodes in it to the paired physical node(s), the amount of physical nodes in service is reduced.

Considered the maturity and advantages of P2P, all the physical nodes are organized as P2P structure. Similar to unstructured P2P, a physical node is joined the P2P network and searches other physical nodes for pair. It has some obvious advantages, including robustness, avoidance of single point failure, etc.

Before virtual node migration, to determine the suitability of the searched physical node, we use the following function to estimate the computation resource utilization after migration:

$$u_{i,e} = \frac{u_i * C_i + u_{i,p} * C_{i,p}}{C_{i,p}} \tag{3}$$

In this function, u_i represents the utilization of computation Resource i of the migration virtual node, C_i represents the capacity of its Resource i, $u_{i,p}$ represents the utilization of computation Resource i of the searched physical node and $C_{i,p}$ represents the capacity of its Resource i. From this function, the computation resource utilization after virtual node migration can be estimated.

Through the computation resource utilization estimation, the load type of the searched physical node after virtual node migration can be estimated by the rule above. If the estimating load type is not overloaded, the searched physical node is selected as a suitable one and the virtual nodes will be migrated to it. Otherwise, PA will try to search another node. If none of the physical nodes within the P2P network of physical nodes can be paired, the algorithm will be cancelled.

4 Experiment and Evaluations

4.1 Experiment Configuration

To demonstrate the effectiveness of the proposed framework, we build an experiment environment with 4 physical nodes and 15 virtual nodes.

In the physical nodes, XenServer is set up as the virtualization server. And in each physical node, WABRM is deployed and interact with the API of XenServer.

In the virtual nodes, Ubuntu 12.04 Server is installed in each node. And Swift storage nodes are deployed in these virtual nodes.

The details of the physical machines used to set up the experiment environment are as follows. 4 physical machines with Intel i5 3.30GHz CPU, 4GB memory and 500GB disk are used as virtualization servers. There are another 4 physical machines with Intel i3 3.30GHz CPU, 4GB memory and 500GB disk. 2 of them are used as proxy nodes while the others are used as clients.

The details of the values of the parameters mentioned above for work-load analysis are as follows. To physical nodes, opts of CPU, memory and NetworkIO are 0.6, 0.95 and 0.95. And weights of CPU, memory and NetworkIO are 0.1, 0.2 and 0.2. Threshold of underloaded is 0.3. To virtual nodes, opts of CPU and memory are 0.7and 0.95. And weights of CPU and memory are 0.2 and 0.8. Threshold of underloaded is 0.3. The choice of the parameters should weight the costs and performance. Presently, we choose the parameters according to the performance priority principle to guarantee the storage system performance. And the method for determining the values of parameters will be presented by the later work.

Physical machine Clinet1 and Client2 are used to simulate clients for data access. 200 users are simulated by them respectively. At the beginning of the experiment,

there is no user access to the storage system. It lasts for 1000s and the work-load of the entire system is quite light in the first 1000s. Then, the simulation of user access starts. The number of concurrent simulated users increases evenly in the first 1000s. Then it reaches steady state. The simulated users send their data requests continuously and the interval between two requests is 100ms. The state lasts for 5000s then the simulation stops.

4.2 Improvement of Computation Resource Utilization

As Figure 4 shown, the number of physical nodes is changed over the work-load of the entire system. At the beginning, since there is no user access to the system, the work-load is very light. Therefore, the virtual nodes can be integrated to 3 physical nodes and number of physical nodes in service is reduced from 4 to 3. With the increase of simulated concurrent user access, the work-load is getting heavier and heavier. Consequently, the number of physical nodes is increase from 3 to 4 to guarantee the performance of the entire system. At last, with the end of the simulation, the work-load is light again and the number of physical nodes is reduced from 4 to 3.

4.3 Improvement of Work-Load Balancing in Physical Machines

As Figure 5 shown, during the heavy work-load period, the work-load of P1 is extremely heavy at the beginning because of hotspot data. And its resource utilization exceeds the threshold, as shown in the yellow circle part. Fortunately, with the effect of WABRM, the work-load of the node is reduced. In addition, part of work-load is transferred to P3, another relatively light physical node. By this way, the work-load of the physical machines is balanced.

The work-load of P2 and P4 is displayed by Figure 6. Compared Figure 6 with Figure 5, the computation resource utilization of each physical machine is controlled under its defined threshold, thus the work-load is controlled. It demonstrates the effect of work-load regulation of WABRM.

4.4 Tuning of System Performance

As Figure 7 shown, the response times of Swift with WABRM and without WABRM are different. With WABRM, the response time is less as a whole.

However, WABRM can't always improve the system performance. Since we set the threshold of resource utilization for overload according to the performance priority principle in this experiment, the average response time is shortened. Practically, low resource utilization may cause high costs of hardware resources. And to different service providers, they should find the suitable compromise of the performance and the costs. WABRM provides the parameters for elastic configuration and they are intuitive. Anyhow, WABRM can eliminate the system bottlenecks caused by hotspot data and guarantee the basic performance of the entire system. In a word, WABRM can achieve its goal of tuning of the system performance.

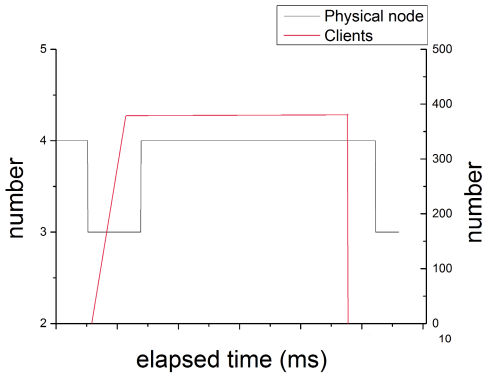

Fig. 4. Change of number of physical nodes over time

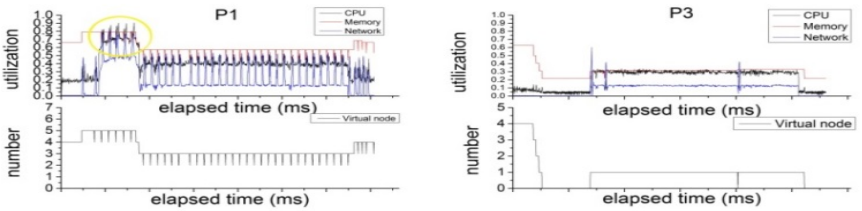

Fig. 5. Effect of work-load balancing

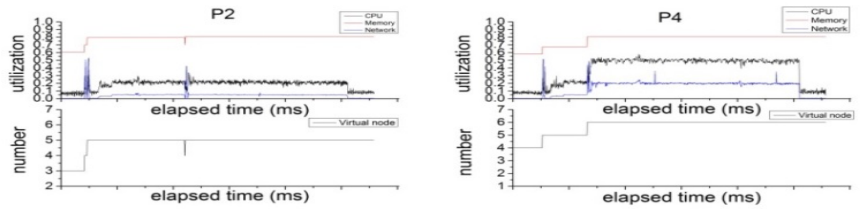

Fig. 6. Work-load of P2 and P4

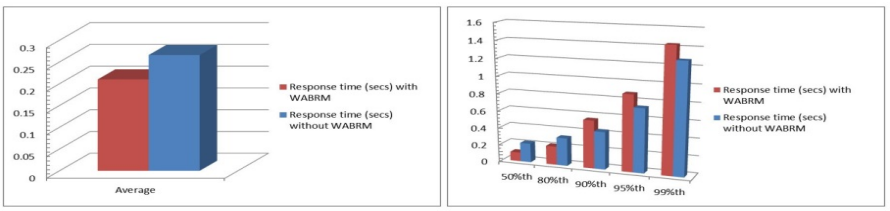

Fig. 7. Comparison of response times (secs) of Swift with and without WABRM

5 Conclusions

As discussed above, dynamic work-load aware balancing should be indispensably complementary to the traditional data amount balancing mechanisms. And WABRM, the proposed framework is effective for tuning system performance when hotspot data exist.

We can draw some conclusions of WABRM. The advantages are as follows. Firstly, WABRM is lightweight and requires no source code modification of the guest OS and storage system. In addition, it provides plenty of interfaces for different implementation of algorithms and environment API. Secondly, WABRM can achieve its design goal of tuning of system performance and improvement of computation resource utilization. Thirdly, through live virtual machine migration, during the regulation operations of WABRM, the service is almost not interrupted.

The disadvantages are as follows. WABRM regulates a node when its work-load exception is detected. However, to prevent the occurrence of the work-load exception, a node's work-load state should be predicted and regulated before the exception. And how to set the values of parameters for work-load analysis remains an unsolved problem. In addition, WABRM is only applied to Swift, in future, we will apply WABRM onto other systems to demonstrate its ubiquitous effectiveness.

Acknowledgement. This paper is supported by Shanghai Municipal Science and Technology Commission under Grant No.11dz1502500.

References

1. Openstack Swift, `http://docs.openstack.org/developer/swift/`
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T.L., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of ACM Symposium on Operating Systems Principles. ACM Press, New York (2003)
3. XenServer, `http://www.citrix.com/products/xenserver/resources-and-support.html`
4. Yamamoto, H., Maruta, D., Oie, Y.: Replication methods for load balancing on distributed storages in P2P networks. In: International Symposium on Applications and the Internet, pp. 264–271. IEEE Press, New York (2005)
5. Madathil, D.K., Thota, R.B., Paul, P., Xie, T.: A Static Data Placement Strategy towards Perfect Load-Balancing for Distributed Storage Clusters. In: International Symposium on Parallel and Distributed Processing, pp. 1–8. IEEE Press, New York (2008)
6. Deng, Y., Lau, R.: Heat Diffusion Based Dynamic Load Balancing for Distributed Virtual Environments. In: 17th ACM Symposium on Virtual Reality Software and Technology, pp. 203–210. ACM Press, New York (2010)
7. Liu, Y., Wan, Y., Jin, Y.: Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment. In: 7th International Conference on Computer Science & Education, Melbourne, VIC, Australia, pp. 851–854 (2012)
8. MongoDB, `http://www.mongodb.org/`

9. Pearce, O., Gambliny, T., Supinskiy, B., et al.: Quantifying the Effectiveness of Load Balance Algorithms. In: 26th ACM International Conference on Supercomputing, pp. 185–194. ACM Press, New York (2012)
10. Zhu, Y., Yu, Y., Wang, W., et al.: A Balanced Allocation Strategy for File Assignment in Parallel I/O Systems. In: 5th IEEE International Conference on Networking, Architecture and Storage, pp. 257–266. IEEE Press, New York (2010)
11. Bui, T.N., Deng, X., Zrncic, C.M.: An Improved Ant-Based Algorithm for the DegreeConstrained Minimum SpanningTree Problem. J. IEEE Transactions on Evolutionary Computation 16, 266–278 (2012)
12. Qin, X., Zhang, W., Wang, W., et al.: Towards a Cost-Aware Data Migration Approach for Key-Value Stores. In: 2012 IEEE International Conference on Cluster Computing, pp. 551–556. IEEE Press, New York (2012)
13. Liu, Z., Lin, M., Wierman, A., et al.: Greening Geographical Load Balancing. In: Liu, Z., Lin, M., Wierman, A., et al. (eds.) 2011 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 233–244. ACM Press, New York (2011)
14. Lin, M., Wierman, A., Andrew, L.L.H., et al.: Dynamic Right-sizing for Powerproportional Data Centers. In: 2011 IEEE INFOCOM, pp. 1098–1106. IEEE Press, New York (2011)
15. Pylot, `http://www.pylot.org/`