2014

ISSN : 0974 - 7435 Volume 10 Issue 16

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(16), 2014 [9360-9369]

The dynamic load balancing protocol in RAMCloud

Changtian Ying[1*], Jiong Yu[2], Chen Bian[1,3]

[1]School of Information Science and Engineering, Xinjiang University, Urumqi, 830046, (CHINA)

[2]School of Software, Xinjiang University, Urumqi, 830046, (CHINA)

[3]School of Information Engineering, Urumqi Vocational University, Urumqi, 830002, (CHINA)

E-mail: yingct@xju.edu.cn

ABSTRACT

The emergence of RAMCloud has improved user experience of online data intensive applications. But due to its random storage policy without the load balancing strategy, in the practical application of large web sites, when faced with highly concurrent access, it may crash server, even make the system paralysis. Therefore, We proposed a load balancing strategy for master server in RAMCloud, which called the dynamic load balancing protocol, it classify the servers into M_client, M_server and M_balance. M_client communicate with M_server through the message of the protocol, and perform load migration. The experiments show that the load balancing strategy can help RAMCloud to deal with random load in a balancing way with sacrificing low latency.

KEYWORDS

RAMCloud; Load balancing strategy; Dynamic behavior; Master server.

INTRODUCTION

Nowadays, large-scale online data intensive applications[1] have found Dynamic Random Access Memory (DRAM) indispensable to meet their performance goals, such as memcached, Redis, RAMCloud, Spark. Both Google and Yahoo! keep their entire Web search indexes in DRAM; Facebook offloads its database servers by caching tens of terabytes of data in DRAM with memcached[2]; and Bigtable allows entire column families to be loaded into memory[3]. Typically these systems depend on high cache hit rates to meet their performance requirements; if caches are flushed, the system may perform so poorly that it is essentially unusable until the cache has refilled.

Unlike Facebook and Bigtable, RAMCloud doesn't use the DRAM as a cache, but keeps all data in DRAM at all times. In order to improve efficiency, RAMCloud takes use of log structure to store objects, and divides the memory into multiple segment. It can provide 100-1000x higher throughput and lower access latency than disk-based storage[4]. Therefore, it can provide high throughput and low latency for online data intensive applications, such as social networks, information retrieval and e-commerce.

At the same time, online data intensive applications put forward higher command to the system, they need the system's ability to handle high concurrency access. In recent years, the server crush and denial of service due to usage soar happened to many data centers. For example, when encountered four-yearly football world cup, the visits of the Twitter may higher 40% or even 150% than the average visits. During the period of 2010 World Cup football match, Twitter had a high rate of denial of service (10%-20%), while the service response delay also increased[5]. For this kind of application, it's difficult to estimate the peak value of access flow and its arrival time. In the practical application of large web sites, when faced with highly concurrent access, it may crash server, even make the system paralysis.

Therefore, the storage systems should have a good strategy that can dynamically balance the system load to deal with massive concurrent access requests. Unfortunately, load balancing is not considered by the current RAMCloud version. Even there are lots of existing load balancing strategys for the clusters, they are mainly designed for disk-based cluster that can not directly applicable to RAMCloud.

In this paper, after comparing the static load balancing algorithm against random method, we find that both of these two strategy can not satisfy the need of RAMCloud. Then we propose the load balancing strategy adapting to RAMCloud, dynamic load balancing protocol (DLBP) for the master server. The protocol enables the master servers dynamically discover, allocate, and request memory to achieve load balancing. Firstly, classify the server into M_client, M_server or M_balance based on the current load status. Secondly, M_client send DLBP message to communicate with M_server for requesting the load migration. After the four shake process, M_client migrates the load to the M_server. The experiment we conducted shows that the DLBP protocol for RAMCloud can efficiently balance the system load comparing to original RAMCloud, but also brings external overhead.

The rest of the paper is organized as following. Section 2 reviews the related work and the motivates of our strategy for load balancing in RAMCloud. Section 3 describes the architecture of RAMCloud. Section 4 describes the design of proposed load balancing strategy. Section 5 describes the evaluation methodology and provides the results from our evaluations and analyzes the findings. Section 6 concludes the work and discusses the future works.

RELATED WORKS

DRAM has historically been a critical part of storage. Early database systems and operating systems as early as Unix augmented storage performance with page/buffer caching in DRAM. More recently, many web applications have found the caching and buffer management provided by their storage systems to be inadequate and instead rely on external application managed look-aside DRAM caches like memcached and Bigtable. And now interest in main-memory databases has revived, as techniques for scaling storage systems have become commonplace, such as H-Store[6]. Unlike them, RAMCloud store entire datasets in DRAM that span many machines, can provide low latency and high throughput. RAMCloud keeps all data in DRAM all the time, so there are no cache misses. It provides a general-purpose storage system that makes it easy for developers to harness the full performance potential of large-scale DRAM storage.

But the current RAMCloud version does not provide the load balancing features. We know that load balancing strategy is based on a distributed system architecture, and it provides an efficient and transparent way to increase throughput, network data processing capability, the flexibility of the system and availability. Due to load balancing of the distributed cluster is very important to overall system performance, it may affect user experience and availability of the distributed cluster system. So many domestic and foreign research institutions and enterprises are focused in this field[7-10].

The load balancing strategy usually include two kinds. First, At the beginning of uploading the data, allocate the data replicas reasonably. Second, during the execution of system, dynamically adjust the workload depending on the status of each server. Concerned with the first aspect, the researches mainly focused on the disks utility and transportation cost of accessing the data. Concerned the second aspect, mainly cared about load transition and how to redirect the request to other servers to avoid some servers becoming hot node.

There are two reasons that existing load balancing strategies do not applicable to RAMCloud

First, the existing load balancing strategies concerned only about the workload of servers which usually indicates disks, but in RAMCloud, each server contains two components, backup server and master server. Backup server load only

affects the recovery efficiency after server crashed, but master server load balancing affects the total performance of the cluster and user experience. So we should consider more about load balance of the data store in memory and use memory load balancing strategy.

Second, Traditional load balancing mainly focused in concentrated load balancing scheme. It may leads the tasks that collecting server status, computing the most appropriate scheduling approach, and combination to each servers centralized to one node, such as the coordinator in RAMCloud. Once this server crashed, the status of the cluster including each server can not be collected, and the load balancing scheme can not be execution any more until the server recovery.

To solve with these two problems, we proposed our load balancing strategy for RAMCloud

The load balancing strategy concerned about master server. For the master server, it has two kind of strategy: static and dynamic. In the static load balancing strategy, initially assign objects to master servers. It can allocate different size of load according to the performance of the master server, but it can not dynamically adjust according to the load changes. In the dynamically load balancing strategy, when the objects has been store in the master server, with time, the server throughput is different owing to different visit traffic. At this time, it needs to adjust the load on the basis of dynamic load balancing protocol. So, we use dynamical load balancing strategy.

Master server load balancing protocol use distributed strategy through the cluster itself to communicate with each other. We judge the system status based on multi-attribute decision theory, then dynamically adjust load.

RAMCLOUD ARCHITECTURE

The RAMCloud project is based in the Department of Computer Science at Stanford University. RAMCloud is a new class of storage for large-scale datacenter applications. It is a key-value store that keeps all data in DRAM at all times. RAMCloud uses backup copies on disk or flash to make its storage durable and available.

As shown in Figure 1, a RAMCloud cluster consists of a large number of storage servers, each of which has two components: a master, which manages RAMCloud objects in its DRAM and services client requests, and a backup, which stores redundant copies of objects from other masters using its disk. The RAMCloud cluster also contains the coordinator. The coordinator manages configuration information such as the network addresses of the storage servers and the locations of objects. The RAMCloud client library maintains a cache of this information, fetching the mappings for each table the first time it is accessed and renew it when the information is changing. Clients can usually issue storage requests directly to the relevant storage server without involving the coordinator[11,12].

In order to ensure the reliability of data, data of each master server need store in disks of backup servers. RAMCloud uses log structured storage technology[13], data in the master servers consist of the log. File writing process as shown in Figure 2, when the master receives a write request, it adds a new object into memory log and log entries are sent to the backup server at the same time. This information is cached in the backup server, and then directly back to the master, are not written to disk. Master completes the request, and after log data has been received in all backup servers, return completion information to client. When the backup server cache is full, by a large transmission accumulated log data are written to disk, and delete the cache data.

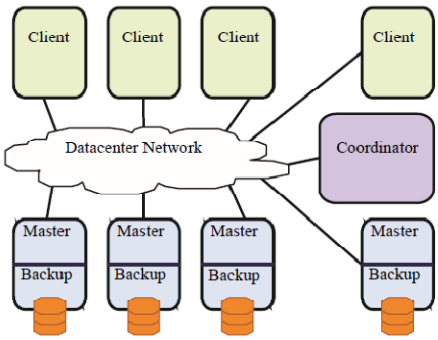


Figure 1: Architecture of RAMCloud

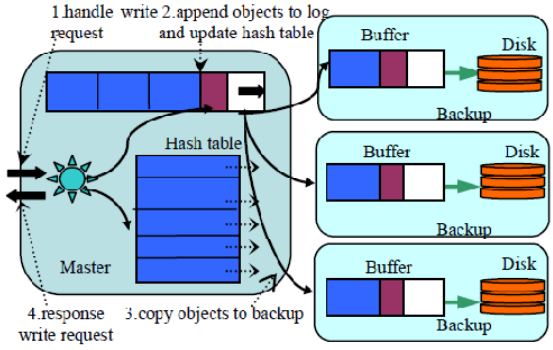


Figure 2: The write process in RAMCloud

Master uses a dynamic tree structure to keep track of the distributed data blocks, and help server to make data segmentation. In RAMCloud, Each master's log is divided into 8MB pieces called segments. The master keeps a count of unused space within each segment, which accumulates as objects are deleted or overwritten. The buffered logging approach allows writes to complete without waiting for disk operations. Each segment is stored in memory, and in order to fault tolerance, in multiple server backup copies (typically two or three). RAMCloud manages object data using a logging approach not only for backup storage, but also for information in DRAM. The hash table is used both to look up objects during storage operations[14].

LOAD BALANCING STRATEGY FOR RAMCLOUD

In this section, after comparing the static load balancing algorithm against random method, we find that both of these two strategy can not satisfy the need of RAMCloud. Then we describe the proposed Load Balancing strategy for RAMCloud. The architecture of RAMCloud load balancing strategy consists of the following components:

Interconnect. The interconnect medium used to link cluster nodes with each other. We conduct our experiment and analyze over Ethernet.

RAMCloud servers. These represent individual computing nodes comprising the cluster. The servers include two components: master servers and backup servers.

Load Balancing Service Manager. The manager, with the proposed protocol and algorithm, is responsible for server status discovery, memory allocation and release. The service manager could be a centralized manager responsible for managing all servers in the cluster, or distributed across all servers. In the initial store of master server and backup server, it use centralized manager coordinator to allocate data. In the process of system operation, with the changes of the visiting load, the master servers dynamically adjust the load, and using a fully distributed dynamic load balancing protocol that does not require centralized control. Each server makes its decision of when, and with whom it shall share[15].

Static load balancing algorithm

In RAMCloud, the system use the hash to map the object to the master server, the hash table is used both to look up objects during storage operations. But it could not allocate data on the basis of performance of server. We proposed Static Load Balancing Algorithm (SLBA) which using the consistency hash method to solve the problem.

(a) Algorithm description

The basic idea of consistent hashing is object and node are mapped to the same hash value space, and use the same hash algorithm. The hash algorithms are usually considered the value mapped to a 64 as the key value, that is, 0 to 2^{64-1} th power of the numerical space; we can imagine this space into a first 0-2^{64-1} phase of the ring. Next use hash function to make the hash calculation of node, the general method can use the IP address of the machine or the machine name as the hash input. Consider the objects, through the hash function calculates the hash value of key out of the distribution of the ring.

SLBA is based on the logarithmic method in literature[16], assigning objects to the heterogeneous master server. Specifically, for M{m_1,..., m_n} server, use the function h_1: $m\rightarrow[0,1]$, $m_i \in$ M mapping to an [0,1] ring. For O{o_1,..., o_m} object, calculate hash value according to h_2:O$\rightarrow$[0,1], $o_j \in$ O. Then assign o_j to the nearest master server according to the distance function:

$$dis(o_j, m_i): \frac{-(\ln(1-\left|h_1(o_j)-h_2(m_i)\right|))}{p_i} \quad (1)$$

This distance function using weights p_i. The value of p_i is the relative computing performance of the master server. The higher performance server is cosidered on the priority, which is likely to store more objects. The SLBA strategy is shown in Algorithm 1.

In this algorithm, the distance function use the weight which representing relative computing ability, thus the load is distributed fairly to the server on the basis of server's performance.

Algorithm 1:Static load balancing algorithm

Input:

Master server: $m_i \in$ M{m_1,…, m_n}
Relative computing performance of master server : $p_i \in$ P{p_1,..., p_n}
object : $o_j \in$ O{o_1,…, o_n}
the distance between object and the nearest server : *shortest*

Output:

the allocation master server : m

1 $i \leftarrow 1$
2 for each m_i in M
3 if $\frac{-(\ln(1-\left|h_1(o_j)-h_2(m_i)\right|))}{p_i} < shortest$

4	then *shortest* $\leftarrow \dfrac{-(\ln(1-\lvert h_1(o_j)-h_2(m_i)\rvert))}{p_i}$
5	$m=m_i$
6	end if
7	increase (i)
8	end for

(b) Experiment evluation

In order to test the feasibility of SLBA, we first implement the SLBA in a simulated environment, comparing to the random method RM.

We simulate the 5 master server, its performance ratio is 1:2:3:4:5, so that in the same time period, the load of each master server is 1:2:3:4:5. We use synthetic load and load size and assume that a system has 1000 objects. Set the object accessing time of each master server are respectively 0.25ms, 0.25/2ms, 0.25/3ms, 0.25/4ms and 0.25/5ms. Using SLBA algorithm assigned the 1000 objects to each master server. Assuming that for each visit to the objects are achieved at the same time, and every 30ms perform SLBA algorithm. We random simulated the arrival time of load, and it ranges among 0~30ms.

Figuure 3 shows the load and total load ratio on each of the master server, when at 300ms after performing 2 algorithms. Using SLBA algorithm, load size and the proportion of the total load is close to the performance of each master server. Using the RM algorithm, load size of each master server are almost the same, the load distribution is unrelated with the performance of master server.

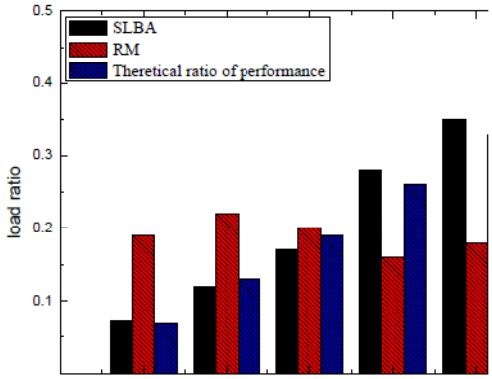
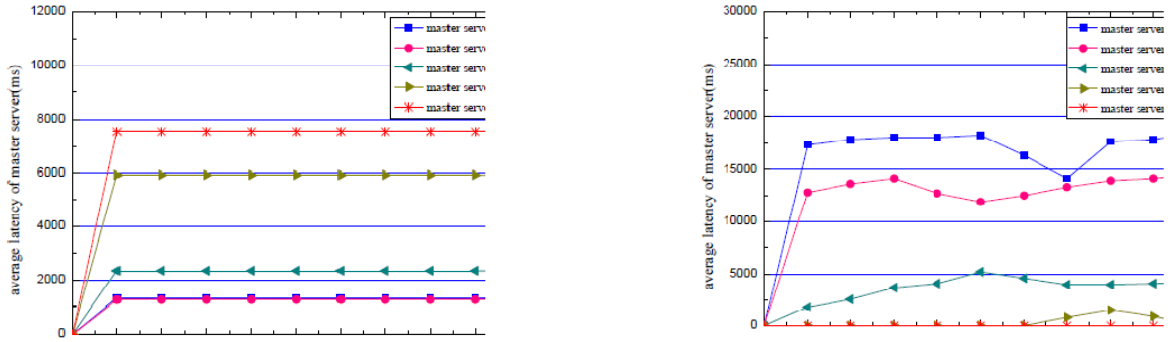


Figure 3: The load ratio of SLBA and RM

Figure 4 shows the average latency of two algorithms. We could see that SLBA distributed the load on the basis of performance of master server in the beginning, but it could not adjusted according to the current access latency, therefore the access latency does not change. RM can not adjust load distribution according to the access latency or the performance of master server, so it could not achieve load balancing by performing RM.



(a) The average latency of SLBA **(b) The average latency of RM**

Figure 4: The average latency of two algorithms

Therefore, we clearly know that static load balancing algorithm can not adapt to the environment. Only if the dynamic load balancing algorithm could adapt to the change of load.

Dynamic load balancing protocol

Then we proposed the Dynamic Load Balancing Protocol (DLBP), which is built on client-server mode. It consists of two components: a protocol for delivering configuration parameters from a M_server to a M_client and a strategy for allocation of memory for M_client. First of all, it should classify the nodes based on the load status. The nodes could be classified to M_client, M_server and M_balance. Then the nodes will adjust the load according to the dynamic load balancing protocol, and migrate data from overloaded node to light load node.

(a) Node classfication algorithm

Static load balancing lacks the desired performance due to typical cluster variations. It is important to dynamically discover, allocate and reclaim the memory adapting to the master servers condition, to optimize the whole cluster performance and energy efficiency.

To achieve the goal, first we classify nodes into three categories according to their server status:

(A) M_client node

a master server that possesses significant amount of free memory and can potentially donate part of its memory space to other memory clients.

(B) M_server node

a master server that is running a high demand application and needs extra memory space.

(C) M_balance node

a self-satisfied master server that has mid-level memory usage that neither offers memory nor needs extra memory.

In general, when the load is smaller than Load_Min (indicating very light load), the master server is classified as an M_server; if load is larger than Load_Max (indicating very heavy load), the master server becomes an M_client; on the other hand, if load stays between Load_Min and Load_Max, the master server is an M_balance node that is self-satisfied.

RAMCloud usually supports for data intensive applications, accordingly the main load pressure of master load server is from read and write operations. Therefore, we could use the access latency to measure the load of the master server. It can also represented by L, the function consists of memory utilization, CPU utilizationand bandwidth utilization, as shown in formula (2):

$$L = k_1 * L(cpu) + k_2 * L(mem) + k_3 * L(bandwidth) \quad \textbf{(2)}$$

Where k_1, k_2, k_3 is the weight parameter, $k_1+k_2+k_3=1$. And L (cpu), L (mem), L ($bandwidth$) represents CPU utilization, memory utilization and bandwidth utilization of a master server respectively.

(b) Dynamic load balancing protocol

During running time, servers are classified into their corresponding category, and engage in the system using the load balancing protocol. The proposed protocol allows servers to exchange information about their memory availability, and facilitate dynamic load balancing decision in a distributed way.

There are five primitives defined for the protocol, as described below.

(A) MEMDISCOVER

This message is generated by the M_client, and the M_client broadcasts the message on its local physical subnet. The MEMDISCOVER message include options that suggest values for the request memory size and its address.

(B) MEMOFFER

M_server sends MEMOFFER message to M_client in response to MEMDISCOVER, and offer with configuration parameters and network address.

(C) MEMREQUEST

This message Generated by an M_client is sent out to a certain M_servers. In this message, the client indicates that it requests load migration. In the case that an M_client has multiple potential M_servers to choose from, the client selects a server based on certain criteria and arbitration strategy, for example, First Come First Serve (FCFS) for simplicity, Round Robin (RR) for fairness, Nearest Client First (NCF) for more energy efficient, etc.

(D) MEMACK

M_server sends MEMACK message to M_client in order to confirm, which concludes the configuration parameters. The message indicates that bind a particular M_server and an M_client at the last stage of shake.

(E) MEMNAK

M_Server sends MEMNAK message to M_client which indicating the denial information. For example, M_server dose not have enough space or performance, so client's request can not be meet now. Or when the load of M_server changes, the status shifts into M_balance, then it should send MEMNAK message to M_client. When M_client receives the message, it could not migrate load to the server.

In order to reduce the overhead of broadcasting, we monitor the M_client during the operation, and limited in a small scope. Such as, in a cluster with more M_client, M_server could broadcast as well.

Take client broadcasting the information as an example, the process is four-time-handshake protocol. As Figure 5 shows, it concludes four steps.

Step 1: The M_Client broadcasts a MEMDISCOVER message in its subnet. The MEMDISCOVER message contains the size of migrate load.

Step 2: Each M_server could use MEMOFFER message to response after receiving the MEMDISCOVER. The MEMOFFER message contains the memory space which could be used. When allocate a memory space to a M_client, M_server should examine whether the space has been used up by other client. If it is not, M_server could send MEMOFFER message to M_client.

Step 3: When M_client receives MEMOFFER message from one or more MEMOFFER, M_client chooses one M_server and send the MEMREQUEST message to this M_server. The message concludes the ip address of M_server and its configuration information. This information is transmitted in the broadcast mode.

Step 4: When M_servers receive the MEMEQUEST from the M_client, it includes two kinds of situations. These unselected M_server receiving the MEMREQUEST indicates that M_client has rejected the service offered by them. The selected M_server receiving the message indicates that M_client will bind to it. This M_server should response to the M_client with MEMACK if it could offer service. The configuration information of MEMACK should not conflict with MEMOFFER. If the selected M_server could not meet the requirements of MEMREQUEST, such as, the requested memory has been allocated, M_server should response with a MEMNAK message.

When M_client receive the MEMACK message, M_client should recheck the parameter and configuration. After such confirmation, M_client can migrate the load to M_server. If M_client receive the MEMNAK message, M_client should restarting the configuration program. M_server access the certain backup server which stores the migrating data to get it. once the data access is complete, the M_server will send the completion information to M_client and coordinator. When the M_client receives it, M_client will delete the corresponding data, and check its status. The coordinator will record the new item for the data.

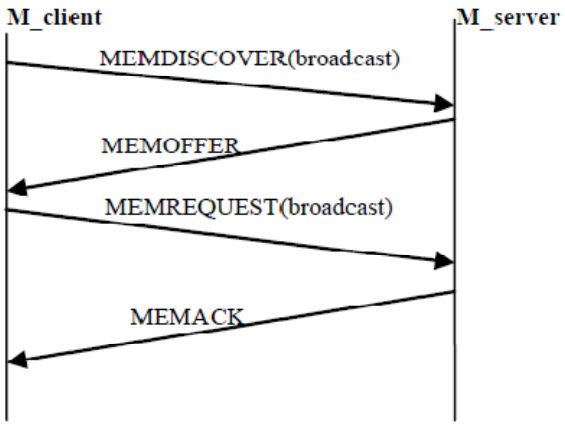


Figure 5: The four handshake of protocol

EXPERIMENT EVALUATION

Experiment environment

In this section, we evaluate the performance of the proposed DLBP comparing to the original RAMCloud without load balancing strategy. The summary is that DLBP can significantly balance the load of the master servers without perceivable performance impact on master servers.

Our experimental setup RAMCloud 1.0 which consists of 6 machines in the cluster, 5 servers, 1 coordinator. And the main code resources to construct the system are accessed from the home page of Standford RAMCloud project[17]. The network topology of the RAMCloud cluster is shown in Figure 6, and the configuration detail of data node server is specified in TABLE 1. The configuration parameters of coordinator is similar to the servers, but it has the redundant power supply. Even though we have two kind of switch, in the experiment, we use ethernet switch to communicate with other master servers, and its performance is less than the Infiniband switch. We know that a server contains two components: master server and backup server. As the TABLE 1 shown, in our experiment, a master server has 4G memory, so the total memory of system is 20G. A backup server has 300GB hard disk space, so the total hard disk space the of system is 1.5TB.

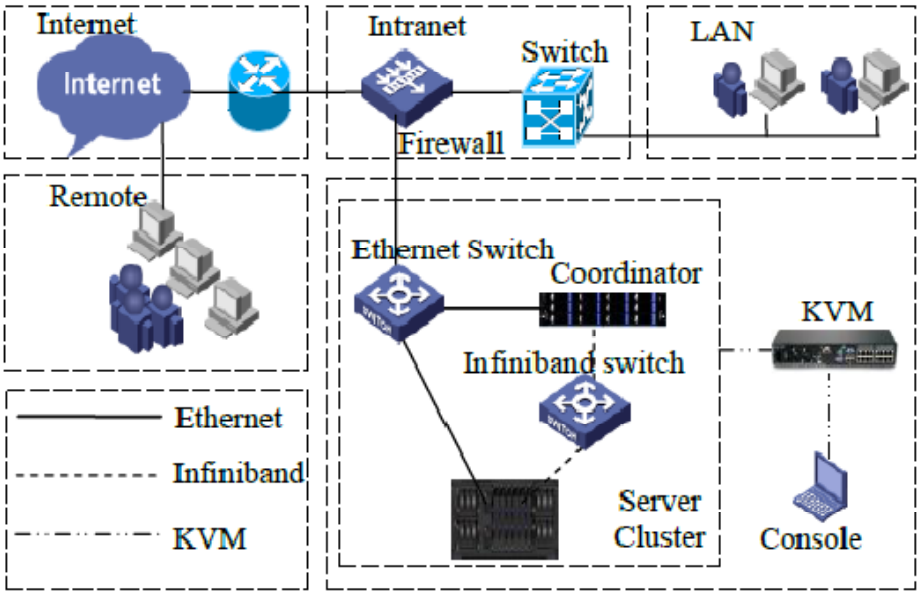


Figure 6: The network topology of cluster

TABLE 1: The parameters of servers

Parameter	Value
CPU	Intel Xeon E5-2670/4 cores/2.6GHz
RAM	4GB/DDR3/1600MHz
OS	Linux Cent OS
Hard disk	300G/2.5 inch/10000rpm/SAS
Infiniband Interface	1*56Gb
Ethernet Interface	2*1000MB

Result analysis

In order to test the performance and system behavior of the DLBP, we have developed the DLBP protocol based on java. To test and analyze the system behavior, we use two kind of datasets which simulate YPCR, BPSR datasets shown in the literature[18]. In this paper, we call two datasets Dataset I and Dataset II. In the initialization stage, upload the datasets randomly to the cluster. Then the parameter measure of load L (*mem*), L (*cpu*) and L (*bandwidth*) is 0.6, 0.25 and 0.15 respectively. We define the server with load between 0-35% as M_server, 36%-65% as M_balance, and 66%-100% as M_server. Then we did the experiments which contains the dynmaic behavior, efficiency and extra overhead of DLBP.

Figure 7 shows the network traffic of two master servers, an M_server node 1, and an M_client node 2. We first uploading the Dataset I and at the 15th second we start uploading the Dataset II. The figure shows the traffic while node 1 is servicing node 2. At around the 5th second, node 2 has heavy load as an M_client and 1 as an M_server, then node 2 request for service and communicates 1 with DLBP messages. Then the load is migrate to 1. At around the 15th second, an application with large memory demand starts on node 1, meanwhile memory demand on node 2 decreases gradually. This run-time change causes 1 to become an M_client and 2 an M_balance. Meanwhile, node 1 starts request with a third node (which is not represent in the figure) with node 1 acting as an M_client. At the same time, node 2 becomes an M_balance that does not request or offer service to other nodes. This visual illustration shows the dynamics and elasticity of the DLBP protocol given the changing memory requirements for running workloads. And the time it takes for a M_client to migrate its data to M_server depends on the network speed, the amount of data and the disk speed.

We can see from Figure 8 that compare load of two strategies in the RAMCloud after uploading Datasets. In the original RAMCloud without load balancing strategy, node 1 and node 2 have heavy load which is bigger than 65%, and node 3 and node 5 has light load which is smaller than 35%, only node 2 is in load balancing status. In the RAMCloud with DLBP, even if the system assigned the load unfair in the beginning, with the time goes by, the DLBP protocol on the master servers will dynamically adjust the load in a balancing way. The reason why original RAMCloud provides less performance improvement compared to DLBP is that, original RAMCloud doesn't have the load balancing algorithm, only randomly assigns objects to the master server, and can not adjust on the basis of load changes.

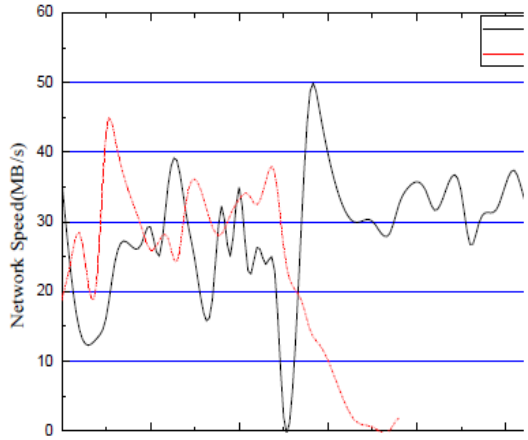


Figure 7: The network traffic of two master servers

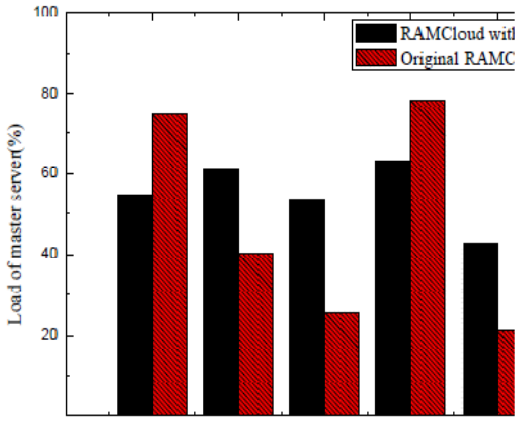


Figure 8: Load comparison of two strategies

Thus far, we have shown that load balancing performance of the protocol. However, this performance improvement comes at the expense of performance degradation for master servers. Because of the overhead induced by DLBP through the network over TCP/IP stack, the latency of DLBP may also increased.

Figure 9, 10 shows the average latency of different number of clients for accessing two datasets which running on master servers. The results show that with the client number increasing, the average access latency of objects are increasing. And the RAMCloud with DLBP incurred more latency than original RAMCloud. Because of two datasets are distributed across several servers, the communication network overhead of DLBP increased the latency, and M_server access data from the disks of backup servers bringing some latency at the same time.

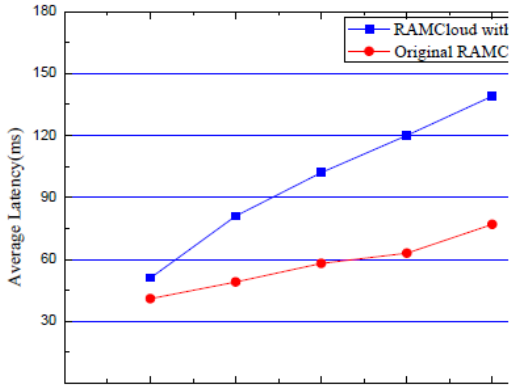


Figure 9: Average latency of Dataset I on two strategies

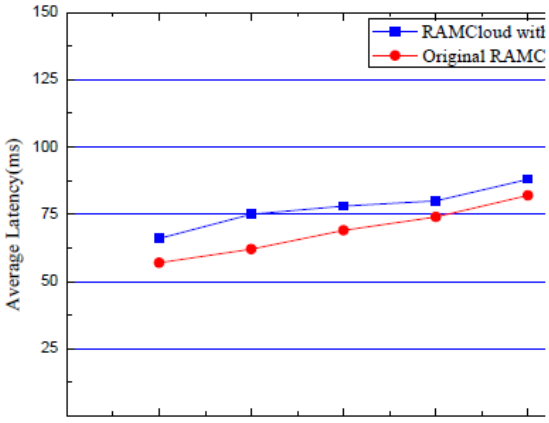


Figure 10: Average latency of Dataset II on two strategies

CONCLUSION

In this paper, we addressed the load balancing problem of master server in RAMCloud. After comparing the static load balancing algorithm against random method, we find that both of these two strategy can not satisfy the need of RAMCloud. Then we proposed the Dynamic Load Balancing Protocol. The DLBP protocol is setup on each server. First, on the basis of the status of load on the server, classified the master server into three kinds: M_client, M_server and M_balance. If the current state of the server is M_client, it should send message to request for migrating data to other servers whose current state is M_server. After the four handshake process, the M_client inform the information of the migrating data to M_server, and M_server access the data from the backup server. After migrating, M_client send migrating information to coordinator, and if it receive the response from the coordinator, it can delete the data which has been migrating to M_server.

We discussed the broadcast-based DLBP protocol, which we implement and evaluate in a small-scale cluster (5 nodes). It can efficiently balance the system according to the changes of load with sacrificing latency. However, when the scale of the system grows to tens, hundreds, or even thousands of nodes, the scalability characteristics must be taken into consideration. Therefore, One interesting direction for future work is to extend RAMCloud cluster to support DLBP protocol performing on a large amount of servers. Another direction is to testify and solve the problems of load balancing caused by server crash and recovery.

ACKNOWLEDGMENT

The work was supported by the Project supported by the National Nature Science Foundation of China (No.61262088,61063042) and the Natural Science Foundation Project of Xinjiang Uygur Autonomous region (2011211A011).

REFERENCES

[1] Y.Zhang, Y.Peng, D.Li; Survery on distributed storage for on line data intensive applications[J/OL], Sciencepaper Online, http://www.paper.edu.cn/releasepaper/content/201303-838 **(2013)**.
[2] Memcached; memcached: a distributed memory object caching system[J/OL], http://www.memcached.org/. **(2013)**.
[3] F.Chang, J.Dean, S.Ghemawat et al.; Bigtable: A Distributed Storage System for Structured Data, In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI'06, CA, USA, 205-218 **(2006)**.
[4] J.Ousterhout, P.Agrawal, D.Erickson et al; The case for RAMCloud[J], Communications of the ACM, **54**, 121-130 **(2011)**.
[5] F.Gu; Research on load balancing problem in distributed file system in cloud computing environment[D], Communication and Information System, Beijing jiaotong university, Beijing, China **(2011)**.
[6] R.Kallman, H.Kimura, J.Natkins et al; H-store: A high-performance, distributed main memory transaction processing system[J], Proceedings of the VLDB Endowment, **1**, 1496-1499 **(2008)**.
[7] H.Wang, Z.Fang, S.Cu; Dynamic sdaptive feedback of load balancing strategy, Journal of Information and Computational Science[J], **8(10)**, 1901-1908 **(2011)**.
[8] B.Dong, X.Li, Q.Wu, L.Xiao et al; A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers[J], Journal of Parallel and Distributed Computing, **72(10)**, 1254-1268 **(2012)**.
[9] H.Chang, Y.Chang, S.Hsiao; Scalable network file systems with load balancing and fault tolerance for web services[J], Journal of Systems and Software, **93**, 102-109 **(2014)**.
[10] K.Dasgupta, B.Mandal, P.Dutta et al; A genetic algorithm based load balancing strategy for cloud computing[J], Procedia Technology, **10**, 340-347 **(2013)**.
[11] R.Stutsman; Durability and crash recovery in distributed in-memory storage systems[D], The department of computer science, Standford, USA **(2013)**.
[12] S.Rumble; Memory and object management in RAMCloud[D], The department of computer science, Standford, USA **(2014)**.
[13] D.Ongaro, S.Rumble, R.Stutsman et al.; Fast crash recovery in RAMCloud[C], Proceedings of the Twenty-third ACM Symposium on Operating Systems Principles. ACM, 29-41 **(2011)**.
[14] S.Rumble, A.Kejriwal, J.Ousterhout; Log-structured memory for DRAM-based storage[C], Proceedings of the 12th USENIX Conference on File and Storage Technologies, Santa Clara, USA **(2014)**.
[15] A.Samih, R.Wang, C.Maciocco et al; Collaborative memories in clusters: opportunities and challenges[J], Transactions on Computational Science XXII Lecture notes in Computer Science, **8360**, 17-41 **(2014)**.
[16] T.Chen, N.Xiao, F.Liu; Adaptive metadata load balancing for object storage systems, Journal of Software, **24(2)**, 331-342 **(2013)**.
[17] RAMCloud Project, The RAMCloud Test Cluster, https://ramcloud.stanford.edu/wiki/display/ramcloud/ RAMCloud.
[18] B.Dong, Q.Zheng, T.Feng et al.; An optimized approach for storing and accessing small files on cloud storage[J], Journal of Network and Computer Applications, **35(6)**, 1847-1862 **(2012)**.