Computing and Informatics, Vol. 38, 2019, 1–18, doi: 10.31577/cai_2019_1_1

A NEW DYNAMIC LOAD BALANCING ALGORITHM FOR MULTI-ROIA

Dong Liu

Department of Computer Science
JiNan University
Guangzhou 510632, China
e-mail: `wze2k@163.com`

Abstract. Real-time Online Interactive Application (ROIA) is an emerging class of large-scale distributed application which can support millions of concurrent users around the world. Due to the dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the dynamic load balancing is a key issue for ROIA. However, most of previous works are dedicated to the load balancing in a single ROIA without considering the variety of different type ROIAs. We take the advantage of differences between ROIAs and propose a new load balancing algorithm for multi-ROIA to improve the scalability of ROIA and increase the resource utilization of system. This paper firstly describes the motivation of the new load balancing algorithm, then presents the dynamic load balancing algorithm for multi-ROIA. Finally, the simulation results are also presented to show the efficiency and feasibility of the new algorithm.

Keywords: ROIA, cloud, MMOG, load balancing, scalability

Mathematics Subject Classification 2010: 68-W99

1 INTRODUCTION

Load balancing is one of the key issues for improving system performance and resource utilization in distributed and parallel computing, and can be divided into two categories: Static Load Balancing (SLB) and Dynamic Load Balancing (DLB). If the load can be determined and divided by a certain method before execution, it belongs to SLB. But, if it can only keep monitoring the system load and dynamically adjusting the load while executing, it belongs to DLB.

Real-time Online Interactive Application (ROIA) [1] is an emerging type of large-scale distributed application. The popular and market-relevant representatives of ROIA are Massively Multi-player Online Game (MMOG), as well as real-time training and e-learning based on high-performance simulation. Therefore, without losing its universality, this paper takes MMOG as a case study of ROIA.

In ROIA, due to dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the loads of computing and communication are difficult to estimate before running. Therefore, SLB strategy is not suitable for ROIA. And many researchers have done some researches on DLB in ROIA. But most of these works are dedicated to DLB in a single ROIA without considering the variety of different ROIAs in some aspects, such as latency tolerance or interaction complexity. The resource utilization of the system and the ability to deal with load peaks still have some limitations. Therefore, ROIA providers have to overprovision their operating infrastructure to cope with the uncertain peak load, which leads to a low and inefficient resource utilization. On the contrary, if ROIA providers do not overprovision the infrastructure, which may acquire high and efficient resource utilization, but also may reduce the ability of the system to deal with an uncertain peak load. To address this problem, we analyze differences between different types of ROIA, and we take an advantage of these differences for complementing each other; we hope that the new system can achieve higher resource utilization and better ability to deal with the peak load.

In previous works [2], we proposed the Multi-ROIA Cloud Platform (MRCP), a new structure to achieve high scalability in ROIA. In this paper, we focus on the corresponding load balancing algorithm of MRCP. We first describe in Section 2 the differences of different type of ROIA, and our new load balancing algorithm is motivated by these differences. In Section 3, we propose the new load balancing algorithm in detail. The new algorithm uses a mixed strategy of the centralized and distributed strategy, and it uses a different strategy in a different layer. Finally, we present experimental results showing that the new system achieves higher resource utilization and better ability to deal with the uncertain peak load.

2 THE MOTIVATION OF NEW LOAD BALANCING ALGORITHM

ROIA contains many types, and the different type of ROIAs have some difference in latency tolerance, interaction complexity and so on. Motivated by these characteristics, this paper proposes a new load balancing algorithm (named Dynamic Load Balancing Algorithm for Multi-ROIA, DLBAM) to improve the scalability of ROIA, to increase the system resource utilization and enhance the ability to cope with load peaks. The specific characteristics are as follows.

2.1 Difference in Latency Tolerance of ROIA

Mark Claypool and other researchers [3, 4] found that different types of ROIA have different sensitivity to network delay because of a different type of a player action. For example, First-Person Shooting (FPS) games are more sensitive to network delay than Real-Time Strategy (RTS) games. With the increase of network delay, the performance of FPS games decline sharply. On the contrary, the performance of RTS games declines slowly with the increasing network delay. So FPS games need obviously a lower network delay to ensure the good user experience than RTS games need.

Based on this characteristic of ROIA, we can deploy a variety of ROIAs on the same hardware platform. If load balancing is needed, the difference in latency tolerance will be considered. The partial load of ROIA which has a low delay sensitive degree in the heavier loaded server will be migrated to other underloaded server. It can improve the resource utilization, and will not cause a great impact on the user experience.

2.2 Difference in Interaction Complexity of ROIA

ROIAs have differences not only in the latency tolerance but also in the frequency and scale of user interaction. Nae et al. [5] used interaction complexity to divide ROIAs by the frequency and scale of user interaction. Assuming that the number of users is n, the interaction complexity may range from $O(n)$ for ROIAs in which users are mostly solitary or the ROIA does not need to make many state changes or compute complex interactions (e.g. puzzle games), to $O(n^2)$ for ROIA in which many users are interacting individually, and to $O(n^3)$ for ROIA in which groups of many players are interacting.

According to this characteristic of ROIA, if we consider the different interaction complexity in the process of dynamic load balance, we can get more satisfactory results for load balancing. That is, when there is a need for load transfer, it should avoid migrating the load of ROIA which has high interaction complexity. Because such migration may lead to producing a new large communication overhead. Therefore the preference should be given to the ROIA which has a low interaction complexity.

2.3 Difference in Load Change of ROIA

Another feature of ROIA is that the variation of load has a certain regularity, and not causing any mess. Although using a single user on ROIA is rather subjective, but the huge number of users makes the load change of ROIA showing a certain regularity. Moreover, due to time zone differences, the use of ROIA in the various regions of the world is not the same. And it makes the differences in load changes of ROIAs depending which servers are located in the various time zone.

Vlad Nae and other researchers selected the popular MMOG (RuneScape [6]) as the object of study. They collected a lot of data of RuneScape servers and statistically analyzed the variation of the simultaneous online users in RuneScape servers. Through the analysis of literature [5], we can see that, on the one hand, the load of ROIA changes obviously, and the difference between upper and lower load peak is huge, so it needs to have good scalability to improve the system resource utilization; on the other hand, generally, the load of ROIA is showing certain rules of change. This is propably affected by the time of day, the load is roughly in accordance with the day cycle up and down fluctuation. Moreover, the load peak is associated with the local time, so that the load peaks of different regions in different time zones generally do not appear in the same time period.

This characteristics is favorable for improving the degree of system resources utilization. If several resource centers are deployed in the different time zones in the world, due to the load peaks of the different resource centers at different times, then the part load can be migrated between the different resource centers.

3 DYNAMIC LOAD BALANCING ALGORITHM FOR MULTI-ROIA

Based on the characteristics of ROIA analyzed above, this section presents a new dynamic load balancing algorithm for multi-ROIA (DLBAM) to enhance the ability to cope with the peak load and to improve the utilization rate of system resources.

3.1 Classification of ROIA and Basic Idea of Algorithm

In order to facilitate the new algorithm, we firstly need to properly classify the ROIAs according to the above mentioned characteristics.

3.1.1 Classification of ROIA

In order to facilitate the load migration, this paper divides ROIA into two categories (named by Dynamic ROIA and Static ROIA) based on the characteristics of ROIA.

Dynamic ROIA refers to the ROIAs which have high latency tolerance and low interaction complexity. Because of the high latency tolerance, it will not impact the user experience when the part load is migrated to another resource center and the delay increased. Furthermore, because of the low interaction complexity, it will not increase the traffic between the two resource centers when the part load migrated to another resource center. So, the part load of dynamic ROIAs is suitable for migration between the resource centers which are located in different places.

Static ROIA refers to the ROIAs which have low latency tolerance and high interaction complexity. Because of the low latency tolerance, it will greatly impact the user experience when the delay is increased. And due to the high interaction complexity, the traffic between the resource centers will greatly increase when the part load is migrated to another resource center. So, this type of ROIA is not suitable for load migration between the resource centers. It is worth noting that the static ROIAs refer to ROIAs not suitable for migration between the resource centers but saying that we are not saying that they cannot be migrated. They also may be migrated between the internal servers in a local resource center.

Table 1 summarizes the classification of ROIA in the new algorithm.

Name	Latency Tolerance	Ideal Delay Threshold	Interaction Complexity
Dynamic ROIA	High	1 000 ms	$O(n)$
Static ROIA	Low	< 500 ms	$> O(n * logn)$

Table 1. Classification of ROIA in DLBAM

3.1.2 Basic Idea of DLBAM

According to the analysis of ROIA's load changes in the above section, the load generally fluctuates by the cycle of one day. And because of the difference of time zone, the emergence times of load peak in different resource center are different. For example, in Figure 1, if there is a load peak in place A, the servers in place C may just have a low peak load. So, it shows a complementary characteristics.

If using the complementary characteristics, each resource center will not need to deploy enough hardware resources to deal with the peak load, it is enough just to deploy appropriate hardware resources. When the high peak load is coming, it can migrate a part of load to the resource center which is in the low peak at that moment.

Moreover, this paper divides ROIA into Dynamic ROIAs and Static ROIAs based on the characteristics of ROIA. The mixing deployment of these two kinds of ROIAs ensures the feasibility of load migrating between resource centers.

According to the idea, this paper presents a new hierarchical balancing algorithm. There are three layers of load balancing in the algorithm. The bottom layer is the load balancing inside of each ROIA in each resource center. The middle layer is the load balancing between ROIAs in each resource center. And the top layer is responsible for the load balancing between the resource centers.

The DLBAM algorithm uses a mixed strategy of the centralized and distributed strategy, and it uses a different strategy in a different layer. In the bottom and middle layer, it uses the centralized strategy, and the distributed strategy is taken in the top layer.

Figure 1 shows the structure of the system using DLBAM. In Figure 1, there are four resource centers deployed in globally distributed four locations. Each resource center has deployed some Dynamic ROIAs and some Static ROIAs at the same time. Each ROIA has a ROIA Scheduler which is responsible for load balancing between the internal servers of each ROIA. In addition, ROIA Scheduler will apply for new resources or release occupied resources to MRCP Local Controller (MLC) according to the load condition of servers in the ROIA. MLC is responsible for the load balancing between the ROIAs in the local resource center and also responsible

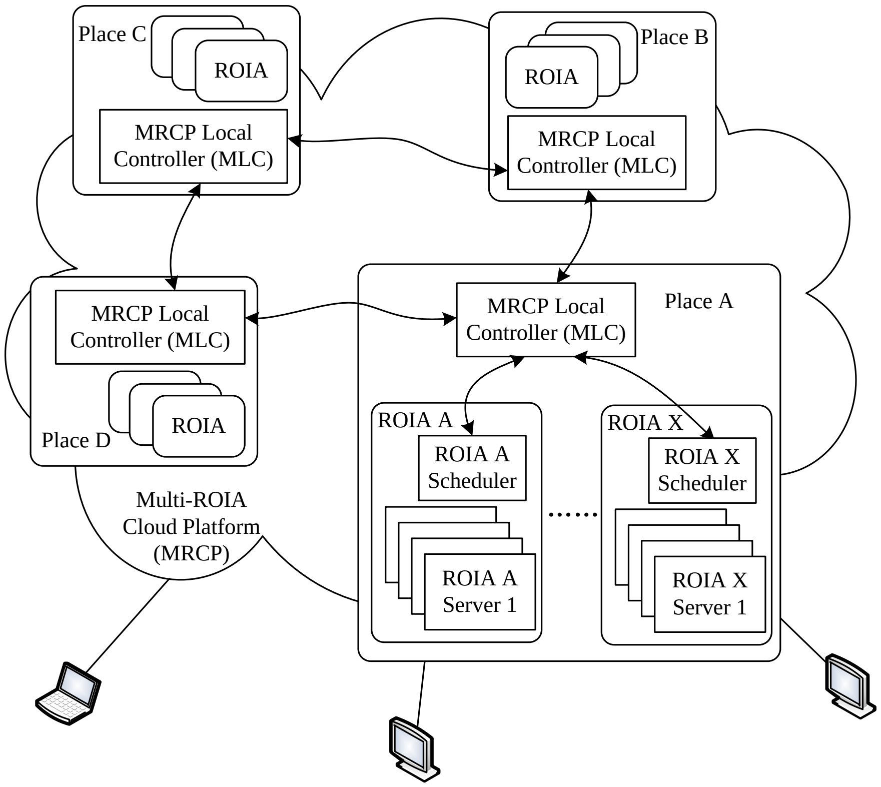

Place C
ROIA
MRCP Local Controller (MLC)
Place B
ROIA
MRCP Local Controller (MLC)
MRCP Local Controller (MLC)
ROIA
Place D
Place A
MRCP Local Controller (MLC)
ROIA A
ROIA A Scheduler
ROIA A Server 1
ROIA X
ROIA X Scheduler
ROIA X Server 1
Multi-ROIA Cloud Platform (MRCP)

Figure 1. Structure diagram of the system using DLBAM

for putting forward the load migration application to the MLC of other resource center when it is necessary.

3.2 Related Definition and Formula

Before introducing the DLBAM algorithm, some variables are defined as follows:

Assumed that there are n ROIAs in the local resource center, and the current number of the k^{th} ROIA's servers is SN_k. The k^{th} ROIA has AN_k avatars and the number of other entities (such as non-player characters, NPC) is represented by EN_k.

The server e of ROIA k is indicated by S_e^k; the avatar i in ROIA k is indicated by a_i^k. And $a_i^k \in S_e^k$ shows that the avatar a_i^k is in the server S_e^k.

$C(a_i^k)$ represents the calculating cost of avatar a_i^k, such as state updating, game logic computing, environment rendering.

$I(a_i^k, a_j^k)$ represents the amount of information exchanged between the avatar a_i^k and a_j^k $(i \neq j)$. When a_i^k and a_j^k are in the same server, the interaction between a_i^k and a_j^k will only increase the computing cost without traffic cost. Only when a_i^k and a_j^k are in the different server, the interaction between them will increase the traffic load.

$V(x)$ represents the computational load increased by the interaction between the avatars in the same server, and $W(x)$ represents the traffic load caused by the interaction between the avatars in the different servers. x is the amount of interaction information.

Let $M_{ROIA}(S_e^k)$ denote the basic amount of memory used for the ROIA server S_e^k under no user links situation. $M_{as}(a_i^k)$ is the amount of memory used for the information of avatar a_i^k. And m_{es} is the amount of memory used for the information of one NPC entity.

The hardware resources which have great impact on the ROIA performance are mainly CPU, network bandwidth and memory. Therefore, the three main aspects needed to be considered when performing load balance: the computational load, traffic load and memory load.

The computational load of server S_e^k can be represented by Equation (1):

$$LC(S_e^k) = \sum_{a_i^k \in S_e^k} C(a_i^k) + \sum_{a_i^k, a_j^k \in S_e^k} V(I(a_i^k, a_j^k)) \qquad (i \neq j). \tag{1}$$

The first part of Equation (1) is the computing cost sum of server S_e^k, such as for state updating, game logic computing, environment rendering, etc. The last part is the sum of computing cost caused by interaction between avatars in the server S_e^k.

The traffic load of server S_e^k can be represented by Equation (2):

$$LN(S_e^k) = \sum_{d=1 \cap d \neq e}^{SN_k} \Big(\sum_{a_i^k \in S_e^k} \sum_{a_j^k \in S_d^k} W(I(a_i^k, a_j^k)) \Big). \tag{2}$$

The memory load of server S_e^k can be represented by Equation (3):

$$LM(S_e^k) = \sum_{i=1}^{AN_k} M_{as}(a_i^k) + EN_k \cdot m_{es} + M_{ROIA}(S_e^k). \tag{3}$$

So, the total load on server can roughly be estimated by Equation (4):

$$L(S_e^k) = LC(S_e^k) + LN(S_e^k) + LM(S_e^k). \tag{4}$$

Assume that $LC_{server}(S_e^k)$, $LN_{server}(S_e^k)$ and $LM_{server}(S_e^k)$ respectively are the max computational load, traffic load and memory load of server S_e^k without degrading the user experience. So, the resource utilization of server S_e^k can be estimated by Equations (5),(6) and (7), respectively.

$$UC(S_e^k) = \frac{LC(S_e^k)}{LC_{server}(S_e^k)}, \tag{5}$$

$$UN(S_e^k) = \frac{LN(S_e^k)}{LN_{server}(S_e^k)}, \tag{6}$$

$$UM(S_e^k) = \frac{LM(S_e^k)}{LM_{server}(S_e^k)}. \tag{7}$$

Let $Threshold_{over}$ denote the server overload threshold. When the resource utilization of the server exceeds this threshold, it shows that the load of the server is too much and asks for load migration. In addition, because dynamic ROIA and static ROIA have a great difference in latency tolerance, therefore they may have different overload threshold.

$$Threshold_{over} = \begin{cases} 1, & \text{dynamic ROIA}, \\ 0.95, & \text{static ROIA}. \end{cases} \tag{8}$$

Let $Threshold_{light}$ denote the underloading threshold. When the server resource utilization is under $Threshold_{light}$, it shows that the server load is too light. To avoid the problem of load jitter between servers, it uses the threshold $Threshold_{top}$ for setting the upper limit of the server resource utilization after load balancing.

3.3 The Internal Load Balancing in Each ROIA

The internal load balancing works on each ROIA scheduler. Each one is responsible for load balancing between the servers of each ROIA. It will provide an application to MRCP Local Controller (MLC) for more hardware resources when the resources are not enough for this ROIA. Also, it will release some resources to MLC when there are excess idle resources in this ROIA.

The internal load balancing of ROIA k roughly shows as follows:

Step 1. Update the *ServerList* which contains the information of servers in ROIA k, compute the resource utilization (denoted by U_i^k) of each server in ROIA k and update the *ServerList* in descending order.

Step 2. Select the first server (S_x^k) from $ServerList$, if the resource utilization (U_x^k) of S_x^k is greater than $Threshold_{Over}$ then continue to the next step, otherwise skip to Step 6.

Step 3. Depending on the load capacity required to migrate, select some appropriate servers from the tail of *ServerList*. And according to Equation (2) to estimate the cost of communication, the servers will be selected in accordance with the size of the communication cost in ascending order to form an alternative destination server list – *CanList*.

Step 4. If *CanList* is empty, the ROIA Scheduler will apply to MRCP Local Controller for resources and jump to Step 6. Otherwise, choose the first server (denoted by S_y^k) from *CanList*, use Equation (4) to estimate the resource utilization of S_y^k (denoted by U_y^k), if U_y^k is greater than $\mathit{Threshold}_{Top}$ then remove S_y^k from *CanList* and repeat Step 4, otherwise continue to the next step.

Step 5. Migrate the load to S_y^k, update U_x^k, U_y^k and *ServerList*, and then jump to Step 2.

Step 6. Select the last server (S_x^k) from *ServerList*, if the resource utilization (U_x^k) is less than $\mathit{Threshold}_{Light}$ then continue to next step, else skip to Step 11.

Step 7. Whether the application is a local application, if it is to continue to the next step, otherwise ready to migrate back to the original resource center, and remove it from *ServerList*, then jump to Step 6.

Step 8. Depending on the load capacity which needs to be migrated, select some appropriate servers from the tail of *ServerList*. And use Equation (2) to estimate the cost of communication, the servers will be selected in accordance with the size of the communication cost in ascending order to form an alternative destination server list – *CanList*.

Step 9. If *CanList* is empty, then jump to Step 11. Otherwise, choose the first server (S_y^k) from *CanList*, use Equation (4) to estimate the resource utilization of S_y^k (U_y^k), if U_y^k greater than $\mathit{Threshold}_{Top}$ then remove S_y^k from *CanList* and repeat Step 9, otherwise continue to the next step.

Step 10. Migrate the load to S_y^k, release the resource of S_x^k, update U_y^k and *ServerList*, then jump to Step 6.

Step 11. Sleep a certain time, and then jump to Step 1.

The algorithm consists of two main parts: Step 2 to Step 5 in algorithm are the processing of overloading, and Step 6 to Step 10 in algorithm are the processing of underloading. The applications in local resource center are divided into two classes: local applications and external applications. The external applications are the applications migrated from other resource center which is overloading. For such applications, the processing of underloading is different with the local applications. The external applications prefer to migrate back to the original resource center, rather than to remain in this local resource center.

3.4 The Load Balancing Between ROIAs and Resource Centers

The load balancing between ROIAs and resource centers works on MRCP Local Controller. It mainly includes the processing of the local resources application, the processing of receiving the response to migration application, and the processing of receiving the application for resources release.

In the system, the applications for local resources can be divided into the following three categories:

1. The resource applications proposed by local static ROIA (*SAppList* represents the queue with such applications)
2. The resource applications proposed by local dynamic ROIA (*DAppList* represents the queue with such applications)
3. The foreign resource applications proposed by other resources center (*InMAppList* represents the queue with such applications)

These three types of resource applications have different priorities when they are processed. To minimize the resource migration between the resource centers the local resource application has its priority. Moreover, due to static ROIA has low latency tolerance, so it will give priority to the resource application proposed by the static ROIA.

In addition, the local resource center may send resource application to another resource center during the peak period. This type of application ensures migrating the load to other resource center, hence named the migration application.

In order to distinguish the above mentioned resource applications, it uses two additional queues: *OSAppList* and *ODAppList*. *OSAppList* stores static ROIA's resource applications which made the system to send migration application but temporarily has not received any response. *ODAppList* stores similar applications of dynamic ROIA.

Therefore, the priority order is as follows:

$$\mathit{OSAppList} > \mathit{SAppList} > \mathit{ODAppList} > \mathit{DAppList} > \mathit{InMAppList}.$$

The processing algorithm for resource applications roughly shows as follows. *FreeList* stores the relevant information of available resource in the local resource center.

Step 1. If $\mathit{SAppList} \neq \emptyset$, select the first resource item (SA_1) from *SAppList* and continue to the next step. Otherwise, jump to Step 4.

Step 2. If $\mathit{FreeList} \neq \emptyset$, then search the appropriate resource (S_x) for SA_1 in *FreeList*. And if found then continue to the next step, else move SA_1 from *SAppList* into *OSAppList*, send migration application and jump to Step 1.

If $\mathit{FreeList} = \emptyset$, move SA_1 from *SAppList* into *OSAppList*, send migration application and jump to Step 10.

Step 3. Assign the resource of (S_x) to SA_1 and delete SA_1 from *SAppList*. Then jump to Step 1.

Step 4. If $\mathit{DAppList} \neq \emptyset$, select the first resource item (DA_1) from *DAppList* and continue to next step. Otherwise, jump to Step 7.

Step 5. If $\mathit{FreeList} \neq \emptyset$, then search the appropriate resource (S_y) for DA_1 in *FreeList*. And if found then continue to the next step, else move DA_1 from *DAppList* into *ODAppList*, send migration application and jump to Step 1.

If *FreeList* $= \emptyset$, move DA_1 from *DAppList* into *ODAppList*, send migration application and jump to Step 10.

Step 6. Assign the resource of (S_y) to DA_1 and delete DA_1 from *DAppList*. Then jump to Step 1.

Step 7. If *InMAppList* $\neq \emptyset$, select the first item IMA_1 from *InMAppList* and continue to next step. Otherwise, jump to Step 10.

Step 8. If *FreeList* $\neq \emptyset$, then search the appropriate resource (S_z) for IMA_1 in *FreeList*. And if found then continue to the next step, else send message ("no appropriate resource") to the requester, delete IMA_1 from *InMAppList* and jump to Step 1.

If *FreeList* $= \emptyset$, send message ("no appropriate resource") to all the requester of *InMAppList*, delete all from *InMAppList* and jump to Step 10.

Step 9. Send message ("found appropriate resource") to the requester of IMA_1, delete S_z from *FreeList*, delete IMA_1 from *InMAppList* and jump to Step 1.

Step 10. Sleep a certain time, and then jump to Step 1.

In addition to the above-described algorithm, there is the processing of receiving the response to migration application, the processing of receiving the application for resource release, and so on, in the system.

4 SIMULATION AND RESULTS ANALYSIS

This section describes the simulation of DLBAM and the comparative analysis of relevant results.

4.1 Experimental Environment

Experiments using Python 3.0 ran a simulation system of DLBAM and traditional dynamic load balancing algorithm, and designed a simulation environment.

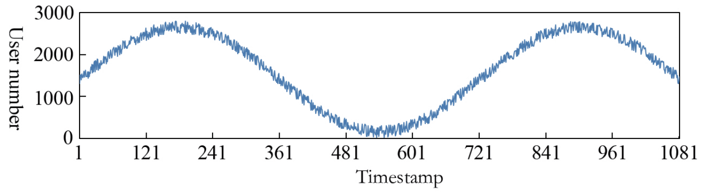
4.1.1 Simulation of Hardware Resources and Applications

In the experiment environment, it has simulated four resource centers in different time zones. Each center has deployed six different ROIA and the proportion of dynamic ROIA and static ROIA is 2 : 1. Each resource center has 180 servers and assumed each ROIA assigned 30 servers initially. For simplicity, the experiment mainly used the number of concurrent online users to simulate the load size. Under the premise to ensure good quality of service, the assumed maximum load for each server is 2 000 online users. If there are more than 2 000 users on one server, the quality of service begin to decline and may even crash in severe cases (but our experiment did not simulate the case of crash).

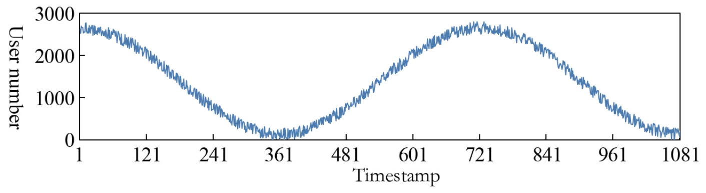
In addition, the traditional algorithm for comparison has the same simulation hardware resource. But in the traditional method, the six ROIAs are deployed independently, and there is no load balancing between the resource centers.

4.1.2 Simulation of Each Server Load

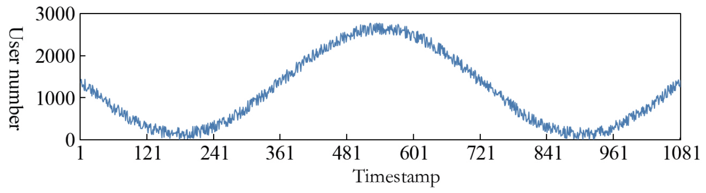
The experiment refers to the actual server load data in reference [5] and produces the fluctuant load data for each server. Moreover, the peak load time of each resource center (four resource centers are numbered by Center 0, Center 1, Center 2 and Center 3) is different. It simulates the impact of different time zones to the load changes.



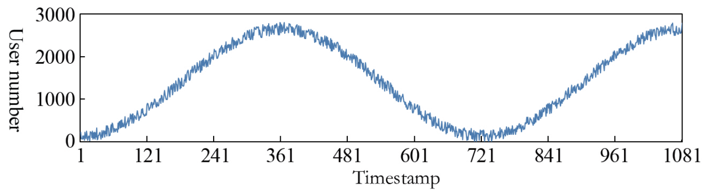
(a) The load changes of server (0,0) in Center 0



(b) The load changes of server (0,0) in Center 1



(c) The load changes of server (0,0) in Center 2



(d) The load changes of server (0,0) in Center 3

Figure 2. The simulation load changes of server $(0, 0)$ in 4 centers

Figure 2 shows the simulated load changes of the server $(0, 0)$ in each resource center. The ordinate unit of Figure 2 is the number of concurrent online users; the horizontal axis unit is the timestamp. The interval between two time points in figure represents two minutes. Figure takes a total of 1081 data points of time, i.e., the figure shows the load variations during 36 hours ($1\,080 * 2/60 = 36$ hours).

4.2 Experimental Results and Analysis

The following results are all based on the simulate load change data which is described in Figure 2. Figure 3 shows the difference between the results of traditional algorithm and DLBAM based on the same server and same load input.

Figure 3 shows the load changes of server $(0, 0)$ in Center 2 during the traditional algorithm and DLBAM running. The blue line represents results of the traditional algorithm run. Due to the peak load of each server on the same resource center appears approximately at the same time and no migration between resource centers in the traditional algorithm, so it happens that no resources are available during the peak period, but too much resources are available during the idle period, as blue line shows. Moreover, the number of users exceeds 2 000 approximately during that time from 400 to 700. According to the pre-set experiments standard, it is indicating that the quality of service begins to decline and may even crash.

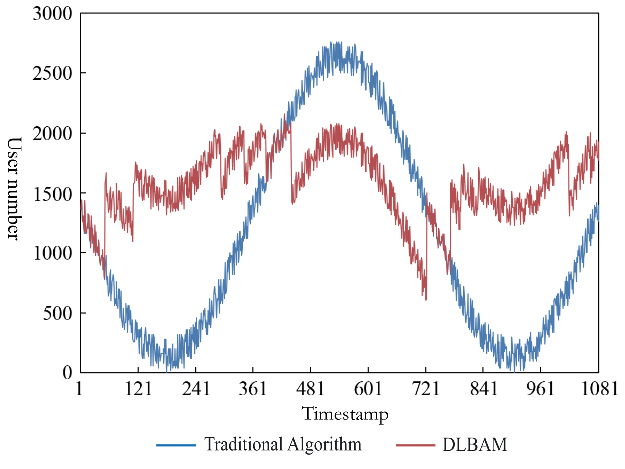


Figure 3. The load changes of server $(0, 0)$ in center 2 during algorithm running

On the contrary, due to having migration between centers in the DBLAM, the user number does not exceed 2 000 during the experiment time, hence the decline of quality of service is avoided. In addition, due to receiving the load of other centers, this server maintains relatively high resource utilization during the low peak period.

Figure 3 shows the difference between the traditional algorithm and DLBAM on one single server and Figure 4 shows the difference from the perspective of entire resource center. In order to facilitate the representation, it takes the ratio of the actual user number to the maximum load as the ordinate (the ratio is multiplied by 100 in Figure 4). The maximum load is assumed having 2 000 online users for each server to ensure good quality of service. The ratio may reflect the resource utilization.

In Figure 4, it shows the ratio changes of each center during algorithm running. The red lines in the figure generally vary between 40 and 100. This indicates that it is not overloaded during the peak period and keeps some suitable load during the low peak period. But the blue lines generally vary between 5 and 130. That means it is overloaded during the peak period and a lot of free resources appears during the low peak period.

From the above analysis, it can be seen that the performance of DLBAM on a single server or on the whole center is better than the performance of traditional algorithm.

5 RELATED WORK

Because some of related work has been introduced in the previous section, only a few other related works about dynamic load balancing are introduced here.

Ren [7] proposed a dynamic load balancing algorithm for cloud computing on the basis of an existing algorithm called WLC (Weighted Least Connection) [8]. WLC assigned a new task based on the number of links on each node. Firstly, it calculated the number of links on each node in the cloud, and then selected the node with minimum links and assigned the task to the node. So WLC algorithm did not consider the other current situation of each node, such as CPU speed, storage capacity and network bandwidth, etc. Ren proposed an improved algorithm called ESWLC (Exponential Smooth Forecast based on Weighted Least Connection). ESWLC algorithm determined whether the node receives a new task after achieving a relative performance of the node, such as CPU power, memory performance, number of links etc.

Mehta and other researchers took a variety of distributed computing environments (such as cloud computing, grid and cluster) into account and proposed WCAP (Workload and Client Aware Policy) [9] based on content-aware dynamic load balancing algorithm. WCAP is a hybrid approach. However, the performance of WCAP in the real distributed environment (e.g. Hadoop) needs a further research verification.

Wang and other researchers proposed a hierarchy load balancing algorithm, called LBMM (Load Balancing Min-Min) [10], based on the OLB (Opportunistic Load Balancing) algorithm [11]. OLB algorithm is a static load balancing strategy for the purpose of keeping each node of cloud with a certain load. It does not consider the execution time of each node, which may lead to slow down of the task processing and also a bottleneck problem may appear. In order to solve these problems, LBMM algorithm takes three-layer architecture. The first layer is Request Manager

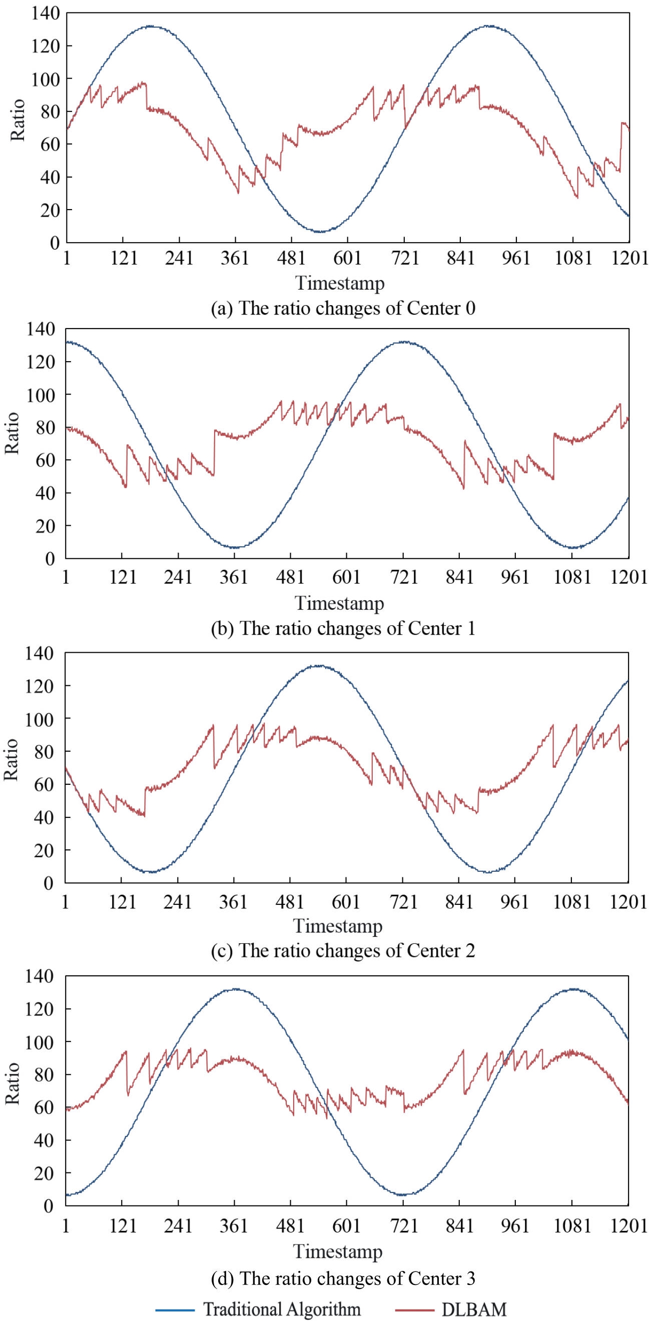


Figure 4. The ratio changes of centers during algorithm running

which is responsible for receiving task requests and assigning the task requests to a Service Manager. When Service Manager receives a task request, it divides it into some subtasks in order to accelerate the processing speed. After the division Service Manager assigns the subtask to service node which is responsible for the execution of subtask.

Bezerra [12] presented a load balancing strategy which uses KD-tree algorithm to dynamically divide the virtual world of ROIA. Kim [13] presented an adaptive load algorithm for solving the problem of traditional graph partitioning method in ROIA. Li [14, 15, 16] and Balogh [17] have done some related works on cloud computing security problems. Nguyen [18] presented a novel development and deployment framework for cloud distributed applications.

In addition, we have also done some preliminary related works on ROIA. Reference [2] proposed a first step of new approach to achieve high scalability in ROIA under cloud environment. And to solve some problem in the traditional Dead Reckoning algorithm, the reference [19] proposed an improved DR algorithm based on target-extrapolation in ROIA.

6 CONCLUSIONS

ROIA is an emerging class of large-scale distributed application which can support millions of concurrent users spread across the world. MMOG is one of the popular and market-relevant representatives of ROIA. Due to the dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the dynamic load balancing is a key issue for ROIA. However, most of the previous works are dedicated to the load balancing in a single ROIA without considering the variety of ROIAs. We take the advantage of differences between ROIAs and propose a new load balancing algorithm for multi-ROIA to improve the scalability of ROIA and increase the resource utilization of the system.

This paper firstly describes the motivation of the new load balancing algorithm. ROIA contains a variety of types, and the different type ROIA has some differences in latency tolerance, interaction complexity, etc. We take an advantage of these characteristics to improve the scalability of ROIA. In Section 3, we present the basic idea of the new algorithm, related definition and formula and the main part of the dynamic load balancing algorithm for multi-ROIA. At the end we describe the simulation of DLBAM and the comparative analyses of relevant results.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61402197).

REFERENCES

[1] Glinka, F.—Raed, A.—Gorlatch, S.—Ploss, A.: A Service-Oriented Interface for Highly Interactive Distributed Application. In: Lin, H. X. et al. (Eds.): Euro-Par 2009 – Parallel Processing Workshops. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6043, 2010, pp. 266–277, doi: 10.1007/978-3-642-14122-5_31.

[2] Liu, D.—Zhao, Y.-L.: A New Approach to Scalable ROIA in Cloud. Proceedings of the 2013 4th Emerging Intelligent Data and Web Technologies (EIDWT), Xi'an, 2013, pp. 51–55, doi: 10.1109/EIDWT.2013.13.

[3] Claypool, M.—Claypool, K.: Latency and Player Actions in Online Games. Communications of the ACM, Vol. 49, 2006, No. 11, pp. 40–45, doi: 10.1145/1167838.1167860.

[4] Claypool, M.: The Effect of Latency on User Performance in Real-Time Strategy Games. Computer Networks, Vol. 49, 2005, No. 1, pp. 52–70, doi: 10.1016/j.comnet.2005.04.008.

[5] Nae, V.—Iosup, A.—Prodan, R.: Dynamic Resource Provisioning in Massively Multiplayer Online Games. IEEE Transaction on Parallel and Distributed Systems, Vol. 22, 2011, No. 3, pp. 380–395, doi: 10.1109/tpds.2010.82.

[6] Jagex Ltd.: RuneScape. `http://www.runescape.com/`, February 2014.

[7] Ren, X.—Lin, R.—Zou, H.: A Dynamic Load Balancing Strategy for Cloud Computing Platform Based on Exponential Smoothing Forecast. Proceedings of International Conference on Cloud Computing and Intelligent Systems (CCIS), Beijing, IEEE, 2011, pp. 220–224, doi: 10.1109/ccis.2011.6045063.

[8] Lee, R.—Jeng, B.: Load-Balancing Tactics in Cloud. Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Beijing, 2011, pp. 447–454, doi: 10.1109/cyberc.2011.79.

[9] Mehta, H.—Kanungo, P.—Chandwani, M.: Decentralized Content Aware Load Balancing Algorithm for Distributed Computing Environments. Proceedings of the 2011 International Conference and Workshop on Emerging Trends in Technology (ICWET '11), Mumbai, 2011, pp. 370–375, doi: 10.1145/1980022.1980102.

[10] Wang, S.-C.—Yan, K.-Q.—Liao, W.-P.—Wang, S.-S.: Towards a Load Balancing in a Three-Level Cloud Computing Network. Proceedings of the 3rd International Conference on Computer Science and Information Technology (ICCSIT), New York, 2010, pp. 108–113, doi: 10.1109/iccsit.2010.5563889.

[11] Sang, A.—Wang, X.—Madihian, M. et al.: Coordinated Load Balancing, Handoff/Cell-Site Selection, and Scheduling in Multi-Cell Packet Data Systems. Wireless Networks, Vol. 14, 2008, No. 1, pp. 103–120, doi: 10.1007/s11276-006-8533-7.

[12] Bezerra, C. E. B.—Comba, J. L. D.—Geyer, C. F. R.: Adaptive Load-Balancing for MMOG Servers Using KD-Trees. Computers in Entertainment, Vol. 10, 2012, No. 3, Art. No. 5, doi: 10.1145/2381876.2381881.

[13] Kim, T.-H.: Adaptive Load Partitioning Algorithm for Massively Multiplayer Online Games. In: Kim, K., Chung, K. Y. (Eds.): IT Convergence and Security 2012. Springer, Dordrecht, Lecture Notes in Electrical Engineering, Vol. 215, 2013, pp. 383–391, doi: 10.1007/978-94-007-5860-5_47.

[14] Li, J.—Wang, Q.—Wang, C.—Cao, N.—Ren, K.—Lou, W.: Fuzzy Keyword Search over Encrypted Data in Cloud Computing. Proceedings of The 29th Conference on Computer Communications (INFOCOM), IEEE, 2010, pp. 441–445, doi: 10.1109/INFCOM.2010.5462196.

[15] Li, J.—Chen, X. F.—Li, M.—Li, J.—Lee, P. P. C.—Lou, W.: Secure Deduplication with Efficient and Reliable Convergent Key Management. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2014, No. 6, pp. 1615–1625, doi: 10.1109/tpds.2013.284.

[16] Li, J.—Chen, X. F.: Efficient Multi-User Keyword Search over Encrypted Data in Cloud Computing. Computing and Informatics, Vol. 32, 2013, No. 4, pp. 723–738.

[17] Balogh, Z.—Gatial, E.—Hluchý, L.—Toegl, R.—Pirker, M.—Hein, D.: Agent-Based Cloud Resource Management for Secure Cloud Infrastructures. Computing and Informatics, Vol. 33, 2014, No. 6, pp. 1333–1355.

[18] Nguyen, B. M.—Tran, V.—Hluchý, L.: A Generic Development and Deployment Framework for Cloud Computing and Distributed Applications. Computing and Informatics, Vol. 32, 2013, No. 3, pp. 461–485.

[19] Liu, D.: An Improved DR Algorithm Based on Target Extrapolating in ROIA Cloud Platform. International Journal of Distributed Sensor Networks, Vol. 9, 2013, No. 12, Art. No. 637328, 8 pp., doi: 10.1155/2013/637328.

Liu Dong works in the Computer Science Department of JiNan University. His research interests include resource management and monitoring in distributed systems.