Evaluation of a Dynamic Data Allocation Method for Web-Based Multi-Server MORPG System

Masaki KOHANA†, Student Member, Shusuke OKAMOTO††b, Masaru KAMADA††c, and Tatsuhiro YONEKURA††d, Members

SUMMARY We have investigated the bottleneck in web-based MORPG system and proposed a load-distribution method using multiple web servers. This technique uses a dynamic data allocation method, called the moving home. This paper describes the evaluation of our method using 4, 8, 16 web servers. We evaluated it on both the single-server system and multi-server system. And we confirm that the effect of the moving home through the comparison between the multi-server system without the moving home and that with the moving home. Our experimental result shows that the upper bound of the number of avatars in the eight-server system with the moving home becomes 380 by contrast that in the single-server system is 200.

key words: web-based application, dynamic data allocation, load distribution

1. Introduction

Recently, due to the spread of broadband access to the Internet and the performance gain of personal computers, a variety of online services are provided. Most services need their special client software. But these client software are bound up with the OS and the architecture. These are also restricted by some security software. A special setting is necessary for these software. This is a hazard for users who do not have the expertise of the computer. It is preferable that their applications do not need the special settings and these operate under a wide variety of environments with few limitations. It is a web-based application that operates with few limitations. Most of web-based applications are not bound with the OS and the architecture. So they can be available if there is a connection to the Internet.

Due to the speeding up JavaScript on web browsers and the development of Ajax technology, web-based applications become to be widespread. Additionally, Ultra Mobile PC (UMPC) becomes popular and FREESPOT is increasing. Hereby, the number of web-based applications has been increasing. However, the access congestion to the web server and lower usability in case of higher frequent communication are major problems on the web-based applications. In addition, the different access latency according to the access point is also a problem.

In this research, we focus on a Multi-Player Online RPG (MORPG) which is built as a web-based application. This type of MORPG needs the web server, which can deal with the frequent request from web browsers. Therefore, the above problems are critical for its performance. The basic technology for these applications is important, because there are many applications like MORPG.

We investigated where the bottleneck of the web-based MORPG is. From the result of the investigation, we consider that a bottleneck in the web-based MORPG is the database access conflict. So, we proposed the dynamic data allocation method for web-based multi-server MORPG system for resolving the bottleneck. We call this method the moving home.

2. Related Work

Ahmed Abdelkhaliek and Angelos Bilas investigated parallelization and scalability of interactive multiplayer game servers [3]. They proposed a threaded game server. Their main challenges were the task decomposition and the synchronization for correct game processing. Their threaded game server can support 25% more players than the sequential game server.

Jeremy Burn et al focused on a network topology [4]. They introduced a notion of critical response time in order to optimize playability and fairness. They formulated an optimal critical response time for server selection problem and solved it for small networks of three different types of network topology. They also introduced an approximate heuristic solution for large networks.

Yi Zhang et al proposed a method named Globally Synchronized Dead-Reckoning with Local Lag, which combines local lag and Globally Synchronized Dead-Reckoning (GS-DR) [5]. This method can eliminate the after-inconsistency and decrease the before-inconsistency. They showed that by combining local lag and GS-DR, the constraint on selecting lag value is removed and a lag, which is smaller than typical network transmission delay, could be used.

J. Waldo discussed the necessity for finding method of making an environment for the online games and virtual worlds [6]. He discussed that the problem solving in a new environment was a great challenge, and a new method of...
the software development for multithreading multi-core distributed system was necessary.

Mejdi Eraslan et al proposed a transport system for a distributed virtual environment with IPv6 [12]. They showed the limitation of the IPv4 and proposed a network architecture using the IPv6 to support the per-packet level QoS for the large-scale virtual environment which needs the great bandwidth. Bart De Vleeschauwer et al proposed a dynamic microcell architecture for virtual environments [13]. To process the higher player density, they divided the virtual world into several microcells and assigned them to a set of servers. And they proposed an algorithm for the optimum microcell assignment to servers. This architecture is important to manage a great number of users.

We can find the several web-based virtual environment, such as RuneScope, Fragoria, and ELEMENTALIA [14]–[16]. And some researchers proposed applications of web-based virtual environments. Erik Champion showed the online exploration of archaeological reconstructions [11]. It uses the JavaScript and XML based application. Helmut Hayer et al proposed the tele-operated laboratory [10]. They provided with the web-based virtual environment and controlled the laboratory experiments via the virtual environment.

We have been studying the virtual environment for MORPGs [1], [2], [8], [9]. Then, we introduced an approach to manage a number of users. This approach is similar to the dynamic microcell architecture which is proposed by Vleeschauwer et al [13]. However, their allocation algorithm previously needs the information of the user density and something to observe the whole of the virtual world. Furthermore, they did not take the web into account. Our approach does not need them. Each web server decides the data allocation to a set of servers with local information. And we use the HTTP connection for the data transfer.

Mark Claypool and Kajal Claypool discussed the precision and the latency that the online game of each category needed [7]. They proposed that the latency which the MORPG needed was 500 msec. We aim adjusting the latency of our MORPG system to 500 msec or less.

3. System Overview

3.1 Game Screen

Figure 1 shows a screenshot of our MORPG system. There are two kinds of characters. The elephant and the frog are avatars of users. Each avatar is operated by the user using mouse and arrow key. The avatar which is located on the center (the elephant) is operated by the user who sees this screen. The other avatar is operated by the other user. The round shaped characters are Non-Player Character (NPC). The NPCs are operated by a program on the web server.

The text box at the upper side is a chat form. To send a message to the other users, the user inputs a message to the chat form and presses the send-button. The user can communicate with the others by their message.

3.2 Game Area and Display Area

Figure 2 shows the relationship between the game area and the display area. The entire part of grid is the total game area. Each part of grids is a block that is the division of the game area. The rectangular with the thick boarder indicates the display area of the web browser. The web server manages the information for characters on each block. The web browser needs only the information on the block included in the display area. Therefore, the web server sends only the information on the blocks, which are in the display area.

4. Single-Server System

4.1 Single-Server Structure

Figure 3 shows the system structure for the single-server system. The web server stores both the background images and the character images. The DB server stores the user information and the character information. The user information includes the password, the login state and so on. The character information includes the character name, the location, the block number and so on.

The web browser sends the request to the web server and receives the character information on the display area.
It is done by JavaScript using Ajax. When the web server receives the request from the web browser, it retrieves the relevant information from the DB server. After retrieving, the web server sends the information to the web browser.

4.2 Implementation

The CGI on the web server receives a request from a web browser and returns the relevant information of characters. These characters are in the block which displayed on the web browser screen. The request includes the name of character, the location of character and the block number. This CGI identifies which the blocks are in the display area. After that, it updates the location of the avatar. It also updates the NPC information occasionally. After updating, it returns the information of the characters on the block which included in the display area.

5. Multi-Server System

5.1 Multi-Server Structure

Figure 4 shows the multi-server structure. Each web browser connects to the nearest web server. All web servers are mutually connected and share the character information. They, however, need the consistent character information because all web browsers connect to the same game area. Then taking synchronization among the web servers becomes an overhead. We introduce the idea of home into the way of managing the character information sharing. This idea assigns ownership of a block to a web server. We call the web server the home of block. Each character is managed on the home of block.

Each web server has two kinds of CGIs (getinst.cgi, getorg.cgi) and two kinds of databases (character.db, home.db). Getinst.cgi is a CGI for web browsers. This CGI reads only the local character information. Getorg.cgi is a CGI for web servers. This CGI is used to share the character information among the web servers.

The character.db stores the character information such as the name of character, the location, the login state of the user, and so on. The home.db stores the information for which web server manages the block. If a web server needs the information of the block managed by the other web server, the getinst.cgi checks the home.db and it finds out which web server is the home of block and it invokes the getorg.cgi on the home of block to get the character information. We call this type of invocation the remote access.

5.2 Remote Access

Figure 5 shows the case of remote access. The web server A manages the left white blocks. The web server B manages the right gray blocks. The web browser which handles the elephant sends the request to the web server A. The getinst.cgi invokes the getorg.cgi on the web server B to get information of the NPC. The getinst.cgi on the web server A receives the information from the web server B and retrieves the local information from the character.db. Finally, it replies to the web browser. The web server B also receives the request from the other web browser which handles the frog. However, the web server B does not invoke the getorg.cgi on the web server A because there are only gray blocks around the frog in the display area.

5.3 Implementation

5.3.1 Getinst.cgi

Figure 6 shows the processing diagram for the getinst.cgi on the multi-server system. This CGI is similar to the getinst.cgi on the single-server system. In the multi-server system, this CGI needs to check the home of the block. If the
web browser needs the block which is managed by the other web server, this CGI invokes the getorg.cgi on the other web server to retrieve the information of the block.

5.3.2 Getorg.cgi

Getorg.cgi is a CGI for web servers. This CGI is used to share the information of the block among the web servers. Figure 7 shows the processing diagram of the getorg.cgi. The getorg.cgi receives the request from the getinst.cgi on the other web server. Like the getinst.cgi, getorg.cgi updates the avatar information and occasionally updates the NPCs information. After updating both the avatar and the NPCs, this CGI retrieves the relevant information from the character.db and replies to the invoker.

6. Moving Home

The getinst.cgi invokes the getorg.cgi on the other web server to retrieve the information of character on the block. However, there is the case that the other web server does not need to manage the block. In this case, the web server which invokes the getorg.cgi prefers to manage the block. Figure 8 shows the case of this situation. The web server A manages the left white blocks, and the web server B manages the right gray blocks. To retrieve the information of the NPC on the gray block, the web server A invokes the getorg.cgi on the web server B. This remote access becomes an overhead. However, there is no web browser which connects to the web server B and needs the character information on the block. So the web server B moves the ownership of the block to the web server A. Then the web server A does not need further remote access for the block. This is the preferable situation. We call the moving the ownership of block “the moving home.”

The transfer data size of the moving home is the almost same as that of the remote access. So, we can implement this mechanism without increasing the transfer data size. Furthermore, we can reduce the amount of transfer data size.

Next, we describe the behavior of system after the moving home. We consider a case that web browser requires the information of the block whose ownership is already moved from the web server B to A. Both the servers know the home of block is already moved from the server B to A. So, the CGI on the server A retrieves the information of the block from the own database, while the CGI on the server B needs the remote access to the server A. The CGI on the other web servers take the remote access to the old home of block, the web server B, because it does not know that the home of block is already moved from the server B to A. Then, the CGI on the server B replies that the host of the new home of block is the web server A. After that, the CGI takes the remote access to the host of the new home of block. Fig-
ure 9 and 10 show the processing diagram of the getinst.cgi and the getorg.cgi for the moving home system respectively. In the moving home system, the getorg.cgi needs to decide whether to move home. The getorg.cgi checks the condition at the state of the checking condition in Fig. 10.

The condition for moving home is whether there is an avatar in the block or not. In order to check it, we put a counter on each block. It counts the number of web browsers which need the information of the block.

The getinst.cgi increments the counter at the setting condition in Fig. 9 when the display area newly includes the block. If the home of block is the other web server, the getinst.cgi invokes the getorg.cgi on the other web server. Then, the getorg.cgi increments the counter of the block at the setting condition in Fig. 10.

The getinst.cgi decrements the counter of the block when the block comes off from the display area. There is no problem in case that everything is in local. We, however, have to deal with a little problem for the remote case. The getinst.cgi can not invoke the getorg.cgi when the home of block is the other web server, because the getinst.cgi needs no information on the block. The counter on the home of block, however, must be decremented by some means or other. So the getinst.cgi invokes the extra CGI to decrement the remote counter. This invoking the extra CGI also becomes a communication overhead.

7. Evaluation

Table 1 shows the specification of the runtime environment for our evaluation. We use four web servers to investigate the basic performance. The entire game area size is 1500 × 1500 pixels and the block size is 100 × 100 pixels. The display area size of the web browser is 300 × 300 pixels. The character image size is 30 × 30 pixels, and the moving velocity is 10 pixels/interval. The update interval of each web browser screen is 1500 msec, so each web browser invokes the getinst.cgi every 1500 msec. The web server sends the information to a user in response to the invoking the CGI. So, the web server sends information to a user every 1500 msec, even if the user does not operate own avatar. The number of NPCs is 100 and the number of avatars is changed from 100 to 300. Each NPC takes a random walk and each avatar moves horizontally. We measured the latency of invoking the getinst.cgi, which includes the transmission delay, the database access delay and the access congestion delay. As we described in Sect. 2, MORPG needs the latency of less than 500 msec for player tolerance [7]. So, we aim to adjust the latency to less than 500 msec using our system. We have built three types of MORPG system for our evaluation, which are the single-server system, the multi-server system without the moving home and the multi-server system with the moving home. We call the second system the static allocation system. And we call the third system the moving home system. We confirm the effect of the static allocation system through the comparison with the single-server system. Similarly, we confirm the effect of the moving home through the comparison with the static allocation system.

7.1 Evaluation for the Static Allocation System

We confirm the advantage of the static allocation system through the comparison with the single-server system. Figure 11 shows the latency of our three types of systems. The upper side graph shows the latency of the single-server system and the middle part graph shows the latency of the static allocation system. The latency of the single-server system is lesser than the static allocation system when the number of avatars is less than 140. However, it exceeds 500 msec when the number of avatars is 200. On the other hand, the latency of the static allocation system is less than 500 msec when the number of avatars is 200, although the latency is larger than the single-server system when the number of avatars is less than 140. As the result, the static allocation system gets the advantage of the performance. However, the latency of the static allocation system exceeds 500 msec when the number of avatars is 220. So, the capacity of the static allocation system is more than that of the single-server system by only 20 avatars.

7.2 Evaluation for the Moving Home

We confirm the effect of the moving home. In Fig. 11, the lower line shows the latency of the moving home system. In the static allocation system, the latency exceeds 500 msec when the number of avatars is more than 220. On the other hand, the latency is less than 500 msec when the number of avatars is 300 in the moving home system. So, the capacity of the moving home system is 300. Figure 12 shows the frequency of the remote access. According to this graph, the moving home system reduces the access latency.
the frequency of the remote access. This means that the communication overhead is reduced.

Figure 13 shows the frequency of the moving home and the remote access in the moving home system. The frequency of the remote access is increasing according to the number of avatars. However, the frequency of the moving home does not increase compared with that of the remote access.

### 7.3 Evaluation of Scalability

To evaluate the scalability of our moving home system, we measured the latencies when the number of web servers is 4, 8, and 16. Figure 14 shows the latency for each number of web servers. The lower line shows the latency of four servers and the middle line shows the that of eight servers. The latency of the four servers is lower than that of the eight servers. However, the latency of the four servers exceeds 500 msec when the number of avatars is 320. The latency of the eight servers exceeds 500 msec when the number of avatars is 380. On the other hand, the latency of the sixteen servers is the worst, although the latency of eight web servers has the higher capacity of the avatar than the that of four web servers. Figure 15 shows the frequency in case that the home of block is already moved to the other web server. The upper line indicates the case of sixteen web servers. It is the most frequent. According to increasing number of the wrong specified host due to moving home, the number of the remote access also increases. This is the reason why the latency of the sixteen servers is the worst.

### 8. Discussion

In our previous work, we found out the major bottleneck of the web based MORPG is the database access conflict. In order to deal with the bottleneck, we introduced the multi-server system and the moving home method. However, as we described in the evaluation section, the latency of the single-server system is lesser than the static allocation system for the small number of avatars. We consider that this is because the communication overhead among web servers in the static allocation system is larger than the database access conflict in the single-server system.

Next, we consider the transfer data size between a browser and a server, and between servers. The transfer data size is about 40 bytes per character. So, for example, if the number of characters is 16 in the display area, the 640 bytes of data are transferred. Additionally, in the static allocation system, the communication between servers is needed. The...
size of transfer data is the same as above. Therefore, the maximum transfer data size of the static allocation system becomes twice that of the single-server system when all the number of blocks are remote. As the result, there are less performance gain although the latency of the static allocation system becomes smaller than the single-server system when the number of avatars is more than 160.

We have compared the static allocation system with the moving home system. The moving home system has a larger capacity of the avatar than the static allocation system by 1.36 times. This is because we can reduce the invoking the getorg.cgi by the moving home. We can also see this result from Fig. 12. The moving home system has the lesser frequency of the remote access than the static allocation system, although the moving home system invokes the extra CGI. As we described at the Sect. 6, the size of transfer data for the moving home is the almost same as the that for the remote access. And we could reduce the frequency of the remote access by the moving home. Therefore, we can also reduce the total communication load by the moving home.

Next, we consider the frequency of the moving home. As shown in Fig. 13, the frequency of the moving home does not increases compared with that of the remote access. This reason is the condition of the moving home. The getorg.cgi moves the home of block when there is no avatar in the block. If the number of avatars increases, the number of blocks with no avatar decreases. This means that the number of blocks whose home can be moved decreases. Therefore, the frequency of the moving home does not increase in contrast to the remote access. As a result, the latency exceeds 500 msec when the number of avatars is 300. If we could apply a more efficient condition, the capacity of avatars would grow more than 300.

We consider the scalability of our moving home system. According to Fig. 14, the more the number of server increases, the more that of remote access increases. And, as Fig. 15 shows, the case that the home is already moved increases according to the number of web servers. This also increases the frequency of the remote accesses to identify the actual home of block. As a result, these degrade scalability. To solve it, we need a mechanism of notification about the moving home to the other web servers.

Finally, we discuss the application with the moving home method. For example, we think our method can apply to the delivery system. This system manages the information of the article of commerce and it’s location. The user can retrieve these information via the web site. However, the database size becomes large if only one database manages the information. We can divide the database according to the area. Each database manages the information which is in the several area. The database that manages a article changes by moving the article. This situation of database access is similar to that on our system. So the moving home method can be applied to this delivery system.

9. Future Works

As we described in Sect. 7, the moving home system has the advantage more than the single-server system. However, the moving home system has some problems. This system does not have a broadcast mechanism. Both origin and destination web server know where the new home of block is, when the home of block moves to the other web server. However, the other web servers never know about the new home of block until they request to the old home. If the home of block moves many times, those web servers must trace the moving. This becomes an overhead. We consider that the frequency of the moving home increases according to both the number of blocks and web servers. Furthermore, the path of the moving home becomes complicated. It may become a major overhead. We need to reduce this overhead.

Next, we consider fairness. In our system, the latency differs according to user. To ensure fairness, the difference of the latency of each user should keep small. We need to implement the mechanism which deals with the fairness. One of there approaches is the synchronization among users by a server. Since a web server is able to know the latency for its users, it can adjust the latency to the worst. If the worst is under the some threshold value, the game becomes viable. We should consider what the web server does for the critical process when the slowest latency exceeds the threshold. This consideration probably differs according to the game category.

10. Conclusion

We evaluated our dynamic data allocation method for reducing the communication overhead in the multiple web server MORPG system. In our system, the game field is divided into multiple blocks. The divided blocks are allocated to the web servers. We call the allocated server the home of block. The home of block manages the character information with this block, while the other web server can retrieve the character information from the home of block using the CGI on the home. The allocation of blocks is changed dynamically by means of the moving home method. The moving home reduces the frequency of the remote access. We confirm that the moving home method raises the capacity of the eight-server system to 380 avatars, while the capacity of the single-server system is 200 avatars. We will also improve the condition of the moving home to increase the avatar capacity.

Acknowledgements

This work was supported by MEXT. Grant-in-Aid for Building Strategic Research Infrastructures.

References


Masaki Kohana was born in 1984. He received the B.Eng., M.Eng. degrees from Seikei University, in 2008 and 2010. He is currently a doctor course student of Graduate School of Science and Technology, Seikei University. His current research interests is load-distribution system for web-based applications.

Shusuke Okamoto was born in 1965. He received the B.Eng., M.Eng. and Dr. Eng. degrees from Seikei University, in 1989, 1991 and 1994. In 1994 he joined the University of Electro-Communications as a research associate. From 1997 to 1999, he was an Assistant Professor at the Graduate School of Information Systems, the University of Electro-Communications. From 1999 to 2005, he was an Assistant Professor at the Department of Computer and Information Sciences, Ibaraki University. He is currently an Associate Professor at Faculty of Science and Technology, Seikei University. His current research interests are programming environments, parallel processing and computer architectures. Dr. Okamoto is a member of IEEE, ACM, IPSJ, and JSSST.

Masaru Kamada was born in Hitachi, Ibaraki, Japan in 1962. He received his bachelor’s (1984), master’s (1986) and doctoral (1988) degrees in engineering from the University of Tsukuba and worked for the same university as a faculty member. In 1992, he joined Ibaraki University where he is currently a professor of computer science. He has been a post-doc fellow at Mathematische Systemtheorie of ETH Zurich in 1993-1995. He served the engineering science society of IEICE as an associate editor of its Japanese transactions (1997–2000), publications secretary (1998–1999) and webmaster (1999). He has been serving Sampling Theory in Signal and Image Processing as its secretary since 2003 and he has been an associate editor of the IEEE Transactions on Industrial Electronics since 2009. Besides web-based services, he is working also on signal and image processing. Dr. Kamada is a member of IEEE and EURASIP.

Tatsuhiro Yonekura was born in 1954. He received his B.S. and M.S. from Nagoya University in 1979 and 1981 respectively. He then joined Yamatake Honeywell, working on development of a distributed computer control system till 1990. He received his Ph.D. Degree in Information Science from the Nagoya University in 1991. He joined Ibaraki University in 1991. He is a Professor at Department of Computer and Information Sciences at Ibaraki University. He is a Professional Member of ACM, IEICE (The Inst. of Electronics, Information and Communication Engineers), BMA (Business Model Association) and VRSJ (Virtual Reality Society of Japan). His research interests are in the area of Human Communication on Internet, E-business, E-society and Distributed Virtual Environment.