Adaptive Web Server using self-reproducing approach

Hiroaki Fukuda
Graduate school of science and technology Keio University
3-14-1 Hiyoshi Kohoku
Yokohama Kanagawa Japan
81-45-563-3925
hiroaki@yy.cs.keio.ac.jp

Kazuyoshi Yamamoto
Graduate school of science and technology Keio University
3-14-1 Hiyoshi Kohoku
Yokohama Kanagawa Japan
81-45-563-3925
yama@yy.cs.keio.ac.jp

Takashi Iijima
Graduate school of science and technology Keio University
3-14-1 Hiyoshi Kohoku
Yokohama Kanagawa Japan
81-45-563-3925
iijima@ae.keio.ac.jp

ABSTRACT
The Internet becomes common for our life. Especially, we usually use web to retrieve some information we want and we can also connect to the Internet using not only some traditional computers but also some mobile devices. Almost all services in the Internet consist of communications between some servers and clients. Since the Internet is wide open and everyone can use it, it is very difficult to estimate the amount of requests and its access patterns. In this paper, we introduce the concept of the self-reproducing and apply to the webservers. In addition, we present the design and implementation of our adaptive webserver system and prove its advantages through some experiments.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms
Reliability

Keywords
Webserver, Load balance, Dynamic, Adaptive

1. INTRODUCTION
The Internet becomes a common way to retrieve some information or get some software we want and we can connect to it with not only desktop style computers but also notebook computers, PDA and handy phones. In addition, the network bandwidth becomes wider, for example, from the dialup connection to ISDN, xDSL and wireless connection. We can use it every time and everywhere. The access number and access pattern from users are not the same. Basically, we usually access some interesting or famous contents a lot and we also access the special event sites, for instance, Olympics or World Cup sites a lot in a short time. Under these situations, the administrators of these sites have to manage them without any problems, for example, the delay of transferring contents or the overload of the servers. Generally, the following approaches are employed in these problems. First, administrators adopt mirror servers that have the same contents as the original server in order to distribute the server load. The second is a proxy server. It caches some contents and transfers them to the users instead of the actual server. However, these approaches are not so effective, because these approaches make the uses configure their own environments. Today, many approaches have been proposed to distribute the server load without the user's operation. In these approaches, administrators usually prepare a fixed number of servers based on their experiences and estimates. Therefore, if the number of client accesses exceeds a predefined limit, the throughput will become worse and in the worst case, the system will go down. However, it is difficult to estimate the amount of client access correctly before starting the system.

Based on these backgrounds, we propose our adaptive webserver system using self-reproducing approach. It can create its mirror servers and distribute the load depend on the client accesses dynamically. Consequently, it is not necessary to prepare some mirror servers before starting the system.

The structure of the remainder of this paper is as follows. In section 2, we describe our approach with contrast to other researches. In section 3, we explain the design and implementation of this system. In section 4 and 5, we show experiments, results and an evaluation. In section 5, we describe some conclusions and future work.

2. APPROACH
The Internet consists of many servers that provide several services and many clients that use the services. We usually don't care how many servers we use and where the servers are deployed when we connect to the Internet, however, we actually use many servers. For instance, in the case of sending or receiving e-mail, we use DNS servers, SMTP servers, and a POP server. The purpose of these servers is to process client requests speedily and correctly. The administrators of them prepare some slave servers and multiplex networks to avoid the overload and a crash of the server. It is clear that a webserver is a main component of several servers that consist of the Internet. We use it with a web browser to retrieve some information. When we send a request to a webserver, it receives and parses the request and returns the requested content. A typical webserver has a lot of configurable parameters to adjust its running policy. An administrator of a webserver configures its parameter before starting it and when additional functionality is required, the usual solution involves shutting down the system, modifying one or more parameters and restarting the system. This procedure doesn't allow the server to be used during upgrade. An Open webserver that can reconfigure its running policies on demand without shutting down is the study to improve this point [2][6]. Otherwise, even if we configure the best parameter and the webserver shows the best performance, a single webserver has its limit. It cannot process a lot of requests beyond its capability.

Today, there are a lot of studies of how to distribute the server load and how to improve the performance because of the access concentration. These studies have little differences between them; however, they can be divided roughly into two types. As shown in Fig. 1, the first one prepares several mirror servers that connect to

each other using a wide range network like the Ethernet. It also prepares the specific hardware to distribute client requests. Clients send a request to that hardware instead of the real webserver and it forwards or redirects it to one of the mirror servers [1][4].

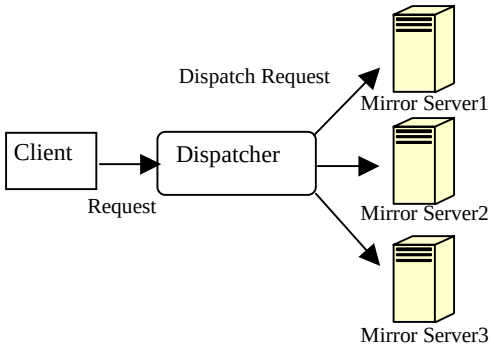

Fig.1 Mirror Server type

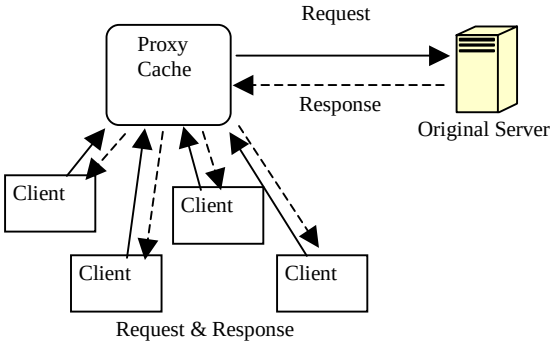

Fig.2 Cache Server type

The second one that is shown in Fig. 2 uses cache servers that save some contents [3][5]. In this case, clients send a request to the cache server instead of the real webserver. After the cache server receives the request, it forwards the request to the real webserver and receives the content. Next, it returns the content and saves it into the caches at the same time. When the other client sends the same request to the cache server, it returns the content from its caches without forwarding the request. These approaches have some problems. In the case of the mirror server approach, the number of the mirror server is fixed and it is difficult to estimate the amount of client access correctly before starting the system in open environment like the Internet. In addition, it costs a lot to prepare the specific hardware to forward client requests and change the IP address or MAC address if necessary. In the case of the cache server approach, clients have to know the name or IP address of a proxy server to use. Moreover, clients have to change their configuration whenever they change their network.

In order to improve these points, we propose the adaptive webserver system introducing the self-reproducing concept. This system can create its mirror server dynamically if the load becomes high. It also deletes its mirror server depending on the state of the server load. We'll describe the design and implementation next.

3. DESIGN AND IMPLEMENTATION

This system consists of RegistServer, AdaptiveWebserver and ClientProxy without any specific hardware. We'll show the detail next

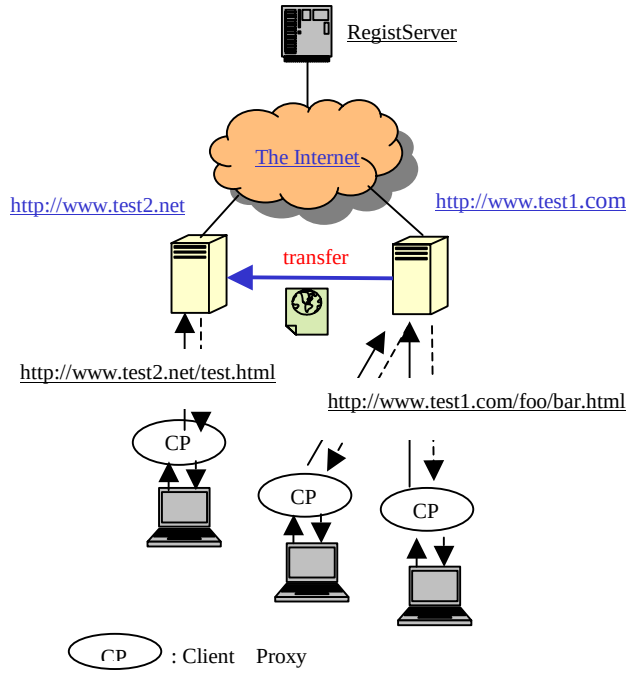

Fig.3 Relationship of each component

- RegistServer

  This is the server that registers the information of AdaptiveWebServer, which is explained later. An AdaptiveWebServer registers itself at this server when it starts. When an AdaptiveWebServer copies its contents, it can get the information of how many servers or which servers are available at that time.

- AdaptiveWebServer

  This server has the same functions as a common webserver, that is, it can parse HTTP1.0 protocol and return the appropriate contents (HTML, GIF, TEXT, etc…) to the clients. It also watches the access frequency of each content and creates the copy of the content that is accessed frequently. In addition, it sends the copied content to the other AdaptiveWebServer. When the AdaptiveWebServer receives the copied content, it manages the content the same as its own. However, if the content is accessed less frequently, it deletes the content. In this way, an AdaptiveWebServer can create its mirror server by copying and transferring the content.

- ClientProxy

  This component works with a client side, that is, it works on the client host that a user uses a browser of. The user who uses this system has to change the browser configuration from the direct connection to the connection via this component. When an AdaptiveWebServer mentioned above

creates the copy of content and transfers it to other AdaptiveWebServer, it has to notify the clients about it. Moreover since this notification has to be transparent for the users, it creates something like a local DNS(Domain Name Service), that is, when it receives a request from a client, it receives the content and the IP address of the mirror server from the real webserver and registers the group of the requested URL and the IP address. In this way, this component can change the mapping of the URL and IP address dynamically.

By using these components, we can develop our adaptive webserver system that can distribute the server load on demand. Next, we show the relationship of these compoments.

As shown Fig. 3, the user, who uses this system, has to set up a ClientProxy on the local host, and change the browser configuration from the direct connection to the connection via the ClientProxy. This configuration is fixed even if the network that the user uses is changed, because the ClientProxy is always deployed on the user's host. This point is different from a common cache server approach. On the other hand, an AdaptiveWebServer has surely the same function of a common webserver, and it also watches the frequency of each content When an AdaptiveWebServer finds the content to be accessed frequently, the server creates a copy of that content and transfers it to another AdaptiveWebServer. In addition, when the server does that action, it needs to get the information of other AdaptiveWebServers. Therefore, it sends a request to the RegistServer, which is mentioned above, to get the information and select the suitable AdaptiveWebServer to transfer the content. Our system can distribute the server load on the fly by the creation or deletion of its mirror server and we explain the detail of creation and deletion of mirror server next.

3.1 Creation of mirror server

An AdaptiveWebServer has some information that is shown below.

1. The access number of each content.
2. The access time of each content
3. The URL list of mirror servers by each content

By using this information, it can find the access number of each content per unit time. When the access number exceeds the threshold, an AdaptiveWebServer begins to copy the content, transfer it and notify the clients. We show this action sequence in Fig. 4 and the behavior of each component below.

At first, (1) "bar.html" that is managed by Host 1 begins to be accessed frequently. (2) Host1 decides that it should copy the content and transfer it, then, Host 1 sends the request for getting the information of other AdaptiveWebServers to the RegistServer if the host has never gotten that information yet, or its expire is over. (3) The RegistServer returns the information with the expire. (4) The host selects another AdaptiveWebServer from several servers using that information. The host that receives the content (Host 2, Host 3)-, saves the content-, and notifies the relative path from its document root to Host 1. Then, Host 1 creates the URL to access the copied content using the relative path and adds it to the content information (5). After that, when a client accesses Host 1 to get the "bar.html", Host1 returns one of the other URLs that it has created instead of the real content. Of course, it sometimes returns the real content even if it has some URLs of the mirror server. As soon as the client receives the URL, it resends a HTTP request to the URL (7). Finally, the client receives the requested content and creates the mapping table of the real URL and the mirror URL. In this way, a client can create the local DNS dynamically. Therefore, if the client tries to get the same content again, it sends the request to a mirror server because it has the mapping table. This operation is transparent for a user because the ClientProxy encapsulates it.

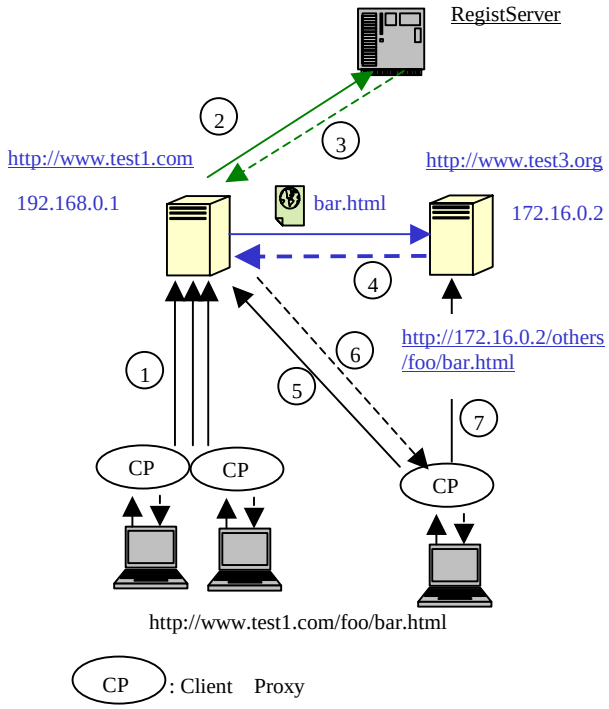

Fig.4 Creation of mirror server

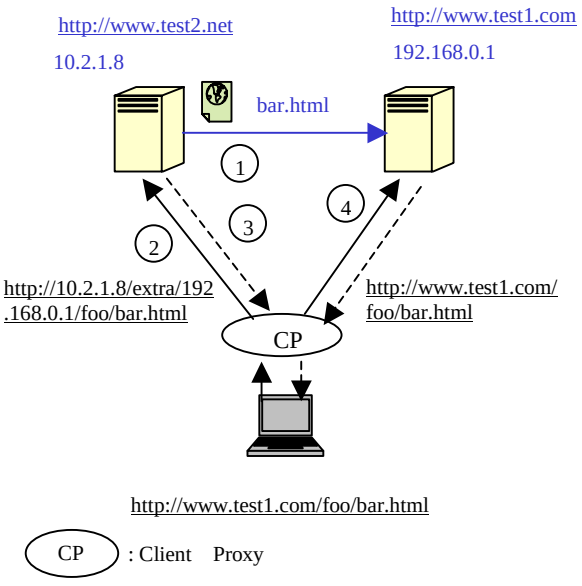

Fig.5 Deletion of mirror server

3.2 Deletion of mirror server

An AdaptiveWebServer deletes the copied content if it isn't accessed frequently. We show the behavior of this deletion below.

In Fig. 5, Host2 received "bar.html" from Host1 and manages it. Host2 decides that it should delete the content because of the access number. Host2 deletes it and notifies this deletion to Host1 (1). When Host1 receives the deletion message, it deletes Host2 from the information of "bar.html". By this operation, Host1 never sends the message that makes clients redirect to Host2. However, the client that has already created the mapping table of Host1 and Host2 still tries to send the request to Host2 (2). In this case, since Host2 has already deleted the requested content, it sends the message that makes the client delete the mapping table to the client instead of the requested content (3). After the client receives the deleted message, it updates the mapping table and resends the request to the real server the get the content. In the same way as the creation of mirror server, this operation is also transparent for a user because of the ClientProxy.

4. EXPERIMENT

This system requires extra CPU and network overhead when it creates a mirror server. In order to confirm the overhead and advantage of this system, we conducted two types of experiments. In type1, many clients request some contents with small size. Besides, in type2, clients download some big size objects. The reason why we conducted two types of experiments is that there are two kinds of bottlenecks about a webserver. One is CPU and another is network. We explain the details next.

4.1 Type1-Experiment

This experiment imitates that many clients requests some small size contents in short time, for example, Olympic website or World Cup website. In these cases, each content size is small but it is requested so frequently.

4.1.1 Environment

In order to conduct this experiment, we prepare six computers and a small content with 100kbytes. We use three computers for AdaptiveWebServer, one for RegistServer and two for client. These are connected to each other by 100Mbps Ethernet. Although we should conduct this experiment using many clients, we create a tester program instead of them. This program can send a HTTP request to an AdaptiveWebServer and parse the received content. It can also parse the message from an AdaptiveWebServer and create the mapping table inside it. The procedure of this experiment is as follows.

1. Start the RegistServer
2. Start three AdaptiveWebServers. As a precondition, each AdaptiveWebServer knows the IP address of the RegistServer. They register their IP address with the RegistServer as soon as they start.
3. Send many requests to a specific URL continuously using ten threads from the tester program in each computer. If the tester receives the redirect message, it creates the mapping table and resends the request to the instructed URL. We measure the transition of the number of accepted request in each AdaptiveWebServer
4. In order to evaluation this system, we also had the same experiment using only one AdaptiveWebServer and measured the transition of the number of accepted request. Thus, the result of more than one AcriveWebServer compares the result of one AdaptiveWebServer. These results are the comparisons of total time of this experiment and the average response time of clients in each case

4.1.2 Result

We show the transition of the number of accepted requests with mirror servers in Fig. 6. The server that has the content of this experiment is the original server (192.168.1.246) and the servers that can receive the copied content are mirror server1 (192.168.1.242), mirror server2 (192.168.2.235). In this figure, the original server creates the copy and transfers it to mirror server1 in one second based on the access number. In the same way, it transfers the copied content to mirror server2 in around three second. The original server also begins to redirect clients to the mirror servers case by case. After this procedure, each server processes about 25-40 client requests in one second and this experiment is finished in about for 49 seconds. In order to compare, it takes about for 71 seconds to process the same requests using only the original server. What is shown in this experiment is that the advantage of this procedure has the value performed even if it has the overhead. Moreover, table 1 shows the comparison of response time per one client. It is clearly shown in this table that a client can get its required contents faster using this system than common webserver. On the other hand, in this experiment, we can't measure the overhead of creating mirror servers. We think the reason is that the content is enough small to prevent from producing an overhead of CUP and network.

	Without mirror server	With mirror server
Response Time (ms)	139	89

Table 1 Response time of each client

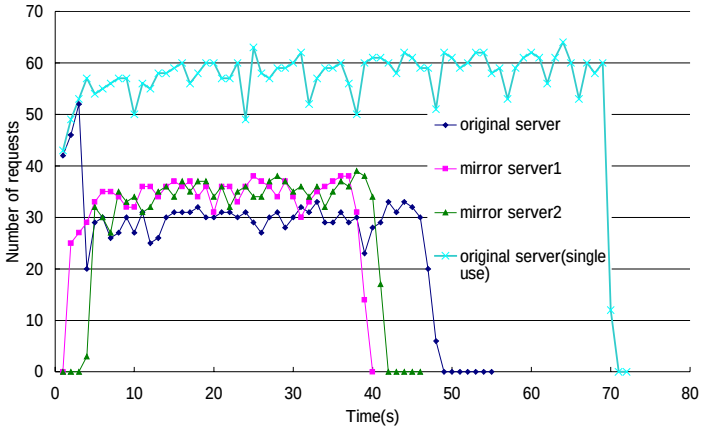

Fig.6 Transition of the number of accepted requests

4.2 Type2-Experiment

This experiment imitates that many clients download some large size contents with HTTP, for example, software download or update. We explain the detail next

4.2.1 Environment

The experiment of type2 is different from type1. We also prepare seven computers and content with 3.3Mbytes. We use two computers for AdaptiveWebServer, one for RegistServer and the rest of them for clients. The computers for AdaptiveWebServer and RegistServer connect to a Layer2 switch with 10Mbps and the rest connect with 100Mbps. On client computers, we create 10 threads to execute the tester program we mentioned before; therefore, each thread can execute one tester program and use 10Mbps. This is the reason why we configure the network bandwidth. On this environment, we measure the download time of some cases explained below.

1. Single AdaptiveWebServer and single client.

   This case is to know the best time of downloading that content.

2. Single AdaptiveWebServer and 40 clients.

   In this case, we start an AdaptiveWebServer and 40 clients simultaneously. We can get the download time that if 40 clients begin to download that content from one webserver.

3. Two AdaptiveWebServer and 40 clients

   In this case, we start two AdaptiveWebServers but the target content is in one AdaptiveWebServer. Then, we start 40 clients simultaneously. When clients start access and many requests accumulate in the queue an AdaptiveWebServer has, the AdaptiveWebServer start to copy and transfer that content.

We measure the average time of downloading that content with copying, transferring and then using mirror server. In addition, we also measure the average time of downloading while the AdaptiveWebServer is doing that procedure of creating a mirror server.

4.2.2 Result

We show the result of this experiment in table2. In case1, it took about 3.8 (s) to download that content with single client from an AdaptiveWebServer. This result is reasonable because an AdaptiveWebServer connects to a Layer2 switch with 10Mbps.

	Download Time(s)
Case1 (single server and single client)	3.8
Case2 (single server and 40 clients)	165.8
Case3 (two server and 40 clients)	73.7
Case3 (while creating mirror server)	197.5

Table 2 Download time in each case

Therefore, if the server can use the whole bandwidth, it will take 2.64 (s). In case2, we believe this result is also reasonable. Because based on the result of case1, it will take 3.8 * 40 = 152 (s). On the other hand, in case3, the average time while the AdaptiveWebServer is creating a mirror server is over the result of case2 for about 31.7 (s). This means that the performance got worse for about 19% while the AdaptiveWebServer was creating a mirror server. However, the average time is below a half of the result of case2 overall.

5. EVALUATION

The results of these experiments show that the system we propose can distribute the server load dynamically without any operation of administrators. Moreover, the overhead produced by creating a copy makes the download time worse only 19% even if the size of content is big. In addition, this disadvantage appears only in creating procedure, then after that, the download time becomes better. Besides, the remarkable point is that while the size of the content clients requests is up to 300kbyte, the main bottleneck of the server is CUP, however, as the size is over 300kbytes, network bandwidth becomes the main bottleneck. We found this when doing type2 experiment. Therefore, in type1, a trigger to create a mirror server is the access number of the target content but in target2, a trigger is the length of the request queue an AdaptiveWebServer has. From these results, this system should be improved to change the policy, which makes a copy with the size of contents and network bandwidth.

6. CONCLUSION

In this paper, we propose our adaptive webserver system using self-reproducing approach in order to distribute the server load dynamically. In addition, we implement this system and confirm the usability in the experiment. This system can distribute the load without any operations of the administrator and its procedure is also transparent for users. Our system doesn't cover dynamic contents like CGI, Servlet, JSP, however, this will perform enough in the case of processing the large size contents such as pictures, movies or music. Furthermore, we would now like to implement the browser that has the function of ClientProxy because in the present implementation, a user has to start a ClientProxy and change the configuration of the browser. This operation is only done once but can be little burden for a user.

7. REFERENCES

[1] V. Cardellini, M. Colajanni, P. Yu.: Dynamic load balancing on web-serve systems, IEEE Internet Computing, May-June 1999

[2] Junichi Suzuki and Yoshikazu Yamamoto. :OpenWebServer: an AdaptiveWeb Server Using Software Patterns, IEEE Communications, Vol. 37, No.4, pages 46-52, April 1999

[3] Barish, G.; Obraczke, K.: World Wide Web caching: trends and tech-niques , IEEE Communications Magazine , Volume: 38 Issue: 5 , May 2000 Page(s): 178 –184

[4] V. Cardellini, M. Colajanni, and P. Yu: DNS Dispatching Algolithms withState Estimators for Scalable Web-Server Clusters, World Wide Web J.Baltzer Science, Bussum, Netherlands, Vol.2, No. 2, July 1999

[5] R. Tewari et al., "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," Techrep. TR98-04, Univ. of Texas at Austin, 1998.