American Journal of Industrial and Business Management, **2013, 3, 349-356**
http://dx.doi.org/10.4236/ajibm.2013.33041 Published Online July 2013 (http://www.scirp.org/journal/ajibm)

A Web Clustered System for Achieving Higher Performances through Load Balancing Mechanism

Young Rhee

Department of Industrial and Management Engineering, Keimyung University, Taegu, South Korea.
Email: yrhee@kmu.ac.kr

Received February 26th, 2013; revised January 20th, 2013; accepted February 20th, 2013

ABSTRACT

In this paper, we study a methodology of distributing client requests in the LVS cluster system. The basic WLC algorithm is studied intensively. A load distributing algorithm which assigns its weight into each real server is devised by considering the state of the network resources on real servers. Network simulation is executed to estimate balanced usage of the resource using web load generating software. In order to analyze the dynamics of server performance due to the workload, we model a system software to evaluate the level of load balancing in the LVS, and perform actual experiments using web agenda. It is shown that the correction potentiality of the suggested algorithm is somewhat better than the WLC algorithm in terms of balanced resources usage.

Keywords: Cluster System; WLC; Correction Potentiality

1. Introduction

Internet traffic has grown exponentially for more than 2 decades along with the popularity of the Internet. Due to the features of the Internet, there are many new and innovative commercial sites attracting several million clients. These commercial sites, such as an online shopping mall, online gaming sites, and online education sites, could potentially lead to a decrease in QoS (Quality of Service) and performances of network in response time by servers [1]. To serve a huge number of client requests with better performance, the administrators of popular websites are facing challenges with the need of improving the capability of the Web servers to meet the satisfactions of the clients.

There are two suggestions for solving these kinds of overload problems of the system. One is to upgrade into higher performing servers, however, it will overload shortly when the number of requests increases. Also, the upgrading procedures can be complex and costly. The other suggestion is incorporating the multi-server system, *i.e.* highly scalable, available and the cost-effective clusters of servers [2].

The architectures of the cluster system may be classified as DNS (Domain Name Server)-based system and dispatcher based system [3]. DNS-based system authorizes DNS server to route a request to a Web cluster in a permitted time interval and imposes fixed logic-name-to-IP mapping outside its time interval. Each server decides whether to fulfill a request, direct to another server, or reject the request. However, the DNS-based system has a downside of controlling only partial of incoming requests in the system. And it is proved that DNS cashing brings in a skewed load on a clustered server by an average of 40% of the total load [3]. An alternative approach to DNS-based architecture aims to achieve full control on client requests and masks the request routing among the Web servers. The dispatcher-based uses simple algorithms for the selection of the Web servers because the dispatcher has to manage all the incoming requests while the amount of processing for each request also has to be maintained at a minimum. To present a better performance for the request, the integration with some sophisticated assignment algorithms would be proposed for the cluster system by having a centralized dispatcher with full control on the incoming requests.

This study focuses on finding a load balancing method to improve its performance efficiency. Numbers of load balancing techniques for the Web server have already been proposed [3]. Some of them solved the problem by adding more hardware to the system, such as Round-Robin DNS [4], others solved the problem by applying software to the controller. The former method may achieve similar effects of load balancing, but it costs. The latter are popularly applied in network devices using control algorithms and achieve fairly good results. Traditional control algorithms for load balancing are Random, RR (Round-Robin), WRR (Weighted Round-Robin), LC (Least Connection), and WLC (Weighted Least Connection). The shortcomings of traditional control algorithms and its arising new challenges form the introduction of new functions on the Web call for the development of more effective new control mechanisms.

In order to analyze the dynamics of server performance due to the workload, we model system software to evaluate the level of load balancing in the LVS (Linux Virtual Server), and perform several experiments using the network applications. The LVS is composed of a cluster system formed with a number of real servers. The LVS control load balancing across the real servers by means of load balancer or director which assigns the incoming requests to real servers based on the certain rules [5]. Some LVS would have multiple directors to improve its system availability. The shortcomings of the existing control mechanism and challenges when introducing a new function on the Web asks for the development of more effective new control mechanisms. In this paper, the general LVS clusters using WLC algorithm are experimented. WLC, which is known to one of the best scheduling algorithms, may have its drawback for maldistribution of incoming requests to the real server pool in terms of the system resources, such as memory used and the number of connections. This may create a more response time for the request. This paper is motivated by the challenges and needs of well distributed control mechanism that can guarantee the performance efficiency. We suggest a control mechanism that calculates and adjusts the weight on real servers, after detecting the system resource periodically. This study therefore develops ideas in finding ways to determine the detecting time interval for the LVS.

The paper is organized as follows. We review related work briefly in Section 2. The design of the proposed load balancing mechanism is presented in Section 3. In Section 4, the performance of the proposed controlled mechanism is evaluated and compared with those of the general LVS cluster using WLC algorithm. Network simulation is executed in Section 5. Finally, Section 6 gives concluding remarks and future works.

2. Related Works

There are several desirable features to implement the distribution of incoming requests among certain sets of servers. In general, these servers will provide the same contents. The scope of this study is not only to describe the detailed features of each distribution techniques, but also refer the reader to the appropriate literature.

2.1. DNS-Based Approaches

DNS-based approaches are implemented by informing different IP addresses whenever clients request the domain name service [3]. Since the IP address selecting in RR or LC manner is not efficient, the better algorithm to solve the inefficiency would be WRR. However, this DNS-based approach has the same limits with simple DNS-based approaches. Because of the hierarchy in DNS, the communication delay among sub DNS makes DNS-based approaches inefficient. Also, heavy burden on DNS is imposed the overhead of cache update among sub DNS, even though TTL of exchange data are set close to zero. DNS-based approach is simply IP-based distributing mechanism and cannot be classified as fundamental settlement to solve the problems.

2.2. Dispatcher-Based Approaches

Dispatcher, a component of a clustered system, receives incoming requests and distributes requests to servers. Request routing among servers is transparent, dealing with addresses at the URL level. The dispatcher, which has a single virtual IP address, uniquely identifies the server in the cluster through a private address that can be at different protocol levels, depending on the architecture. The majority of dispatcher-based architectures use simple algorithms to decide which server will manage incoming requests. Such simple algorithms include random selection, RR, and LC. Even though the dispatcher can achieve well load balancing, it can be a bottleneck with increased requests, and a poor performing dispatcher can cause the whole system to fail.

2.3. Linux Virtual Server

The LVS is highly scalable server built on a cluster of real servers, with the Linux OS based load balancing systems [5]. The architecture of the cluster is transparent to end users. The client only sees a single virtual server. The real servers may be interconnected general network. The load balancer, which distributes requests into different real servers, is located in the front of the real server pool. Scalability is achieved by adding or removing a real server in the pool. The LVS uses traditional scheduling algorithms such as RR or WLC to allocate session connections among real servers.

3. A Proposed Methodology

The LVS is an open software project to provide Linux OS-based load balancing. This cluster-based Web server consists of the load balancer and real servers, and the IP addresses are mapped between these two groups. Exception for the real servers, 2 more IP addresses are required to form this cluster system; one for the virtual server and the other for the load balancer. The request packet destined for a virtual IP address arrives at the load balancer when accessing the service provided by the cluster of the real servers. The load balancer examines the packet's destination address and port number. If they are matched for a virtual server service, a real server is chosen from the clusters by a scheduling algorithm. Then, the destination address and the port of the packets are rewritten as those of the chosen real server, and the packet is forwarded to the real server. When the reply packets come back, the load balancer rewrites the source address and the port of the packets as one of those virtual servers, so that the source addresses always point to the virtual IP address. In this manner, request and response packets need to pass through the load balancer.

The load balancer encapsulates the packet within an IP datagram and forwards it to the chosen real server, when the load balancer sends requests to the real servers through an IP tunnel. As soon as the encapsulated packet arrives, the real server decapsulates and processes the request, and finally returns the response directly to the user. The load balancer sends requests to real servers through an IP tunnel in this clustered Web server.

In this section, WLC algorithm is briefly discussed, and a load balancing algorithm with considering dynamic weight which depends on the state of servers, is presented.

3.1. WLC Scheduling Algorithm

WLC scheduling algorithm basically allocates the requests to the real server with the smallest number of connections. Additionally, WLC algorithm considers each server's capability. It assigns a weight to each server according to its capacity so that the more capable server is ready to receive the more requests. Consequently, the initial assignment to real servers is the server with the biggest weight, and the next assignment is the server that satisfies the following equation.

$$\min\left[\frac{\dfrac{C_i}{C_{all}-conn}}{Wi}\right] = \min\left\{\frac{C_i}{W_i}\right\} \tag{1}$$

$$\text{where } i = 1, 2, 3 \ldots n$$

In (1), $C_{all\text{-}conn}$ and C_i denote the total number of connections and the current number of connections in server i respectively. And W_i also represents the weight for server i. As seen in (1), the same result is obtained when C_i, the current number of connections in server i divided by its corresponding weight, W_i, since $C_{all\text{-}conn}$ is constant applied for all servers.

3.2. Dynamic Load Balancing Mechanism

In LVS cluster system with WLC algorithm, the connections to servers can be preponderant to a server exceptionally, since WLC algorithm only counts the current number of connections without counting the load for each real server. It is known that the performance for the cluster system depends on the poor capable server. Thus, the LVS system typically consists of the same capable servers to overcome this possible performance problem. And the weights for all servers are set to 1 initially.

However, if we have a grip of the system information for each real server, an efficient LVS can be possible based on this information. The system information can be defined as any performance characteristics such as available memory, processor utilization and number of connection established. Among them, the available memory is the only factor we could figure out. The stress of a server in terms of performance counters is tested and analyzed using three types workload, small size but very frequent, large size and various size [5,6]. In this section, we present an efficient LVS cluster system that the weight for each real server is obtained periodically. **Figure 1** gives a demonstration of how LVS works.

Figure 2 is an algorithm that computes weight based on memory's consumption of each collected real server. Basic conception of weight computation is that the differences of available memory estimated from each real server is divided by mean memory per connection, and the result is given to real servers which have available memories as weights, and 1 is given to real servers with relatively low available memory.

$$\text{weight}_{\max} = \frac{(\text{available memory})_{\max} - (\text{available memory})_{\min}}{(\text{mean memory used per connection})} \tag{2}$$

$$\text{weight}_{\min} = 1$$

(2) shows how to computes weight, let say alternative 1. In this method, *weight*$_{\max}$ means weight of a real server with more available memory, and *weight*$_{\min}$ for less available memory respectively. And (*available mem ory*)$_{\max}$

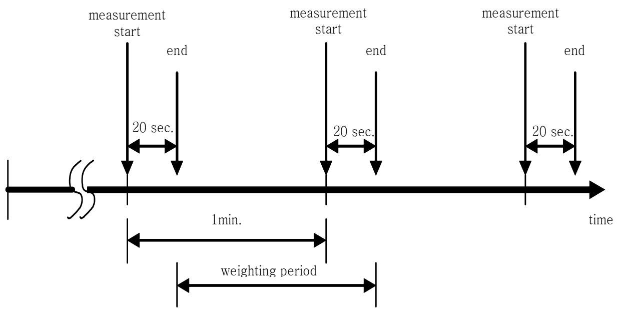

Figure 1. Web load sampling process.

```
push(@wei_a,swap($vs1[0],$vs1[1],$vs1[3],$vs1[5],$weights),swap($vs2[0],$vs2[1],$vs2[3],$vs2[5],$weig
hts));
$min_mem = min_value($vs1[2],$vs2[2]);

if(($wei_a[0] == 0)&&($wei_a[1] == 0))
{
    $wei_a[0] = 1000000;
    $wei_a[1] = 1000000;
}

@c_mem = ([$wei_a[0],$vs1[2],$min_mem],[$wei_a[1],$vs2[2],$min_mem]);
for ($i = 0; $i < 2; $i++)
{
    push(@weight,check_weight_mem($c_mem[$i][0],$c_mem[$i][1],$c_mem[$i][2]));
}

for  ($i = 0; $i < 2; $i++)
{
    $weight[$i] = int($weight[$i] * 1000);
}
    $max_weit = max_value($weight[0],$weight[1]);

if ($max_weit > 65535)
{
    for ($i=0;$i<2;$i++)
    {
        $weight[$i] = int ($weight[$i] * dec_rate($max_weit));
    }
}

system("ipvsadm -e -t 210.107.211.59:80 -r 210.107.211.52 -i -w $weight[0]");
system("ipvsadm -e -t 210.107.211.59:80 -r 210.107.211.53 -i -w $weight[1]");

$dblvs->do("insert into server_weight values(null,null,'$weight[0]','$weight[1]')");
```

Figure 2. Dynamic load balancing algorithm.

and (*available memory*)$_{\min}$ denote maximum unused memory and minimum unused memory of corresponding real servers. *mean memory used per connection* is defined as mean used amount of memory that HTTPD process and PERL process of real server occupies, is used 3500 KByte as an empirical value in this experiment. After computing the weights for real servers, (1) is applied for assigning each connection to real servers.

$$\text{weight}_{\max} = k + \frac{(\text{available memory})_{\max} - (\text{available memory})_{\min}}{(\text{mean memory used per connection})} \quad (3)$$
$$\text{weight}_{\min} = k$$

As can be seen in (3), the second alternative to compute weights for real servers, is simply add constant value *k* to weights in (2). The difference from load distributing alternative 1 is that *k* is given to the real server, not 1, so that one can desensitize the degree of reflection of load distribution more than alternative 1. The value *k* depends on number of access to real server in a minute, lies between 1 and 2000 in this experiment. The reason why we are setting this number is because roughly 4000 requests in a minute from load test are dispersed to 2 real servers.

4. Network Simulation

4.1. Composition of Clustering System

The system in this study is based on LVS-Howto [7] and other references [8]. To compose basic LVS cluster system, which is the basis of study, IP tunneling method is selected as logical topology, and WLC method and two load distributing alternatives are chosen as work assignment algorithm. The experiments are executed under isolated Ethernet environment at night to exclude all the factors that can affect system as much as possible. Laboratory equipments consists of 1 director, 2 real servers, 3 load servers to generate load to LVS cluster system, and 1 monitoring system, and specific information on equipments will be emitted. Following softwares are installed to LVS cluster system and load servers above, and the composition form of system and load server clusters consist of like **Figure 3**.

NT and Windows XP servers are used for load servers with normal options except for WWW and ftp services. Load generator and TestTalk, which generates web load and communicates with load server respectively, are installed in these three servers. In this research, virtual client, which is created by load generator, is connected to LVS cluster system to impose load. Each connection is transferred to the director, and distributed to real server according to work assignment algorithm. Real server processes requests, and transfers the results to the client. The status of available memory is calculated every single minute after booting in each real server itself, and the results are stored in MySQL database in director server. Finally, TestTalk and monitoring softwares are installed to the monitor that observes and controls load test process.

Resources used to see the degree of load of a real server are the amount of available physical memory, the number of processes in uninterruptible sleep status, the number of processes that are swapped out, and so on. The amount of available physical memory is selected as a main parameter. In the case of servers with pre-forking method like Apache, when there is a request from a client, arbitrary numbers of servers are executed and the servers process all requests from clients. If there is no spare server process to manage, the servers increase the number of processes so that they can manage requests. In this procedure, server process usually occupies 2.7 - 3.5 Megabytes and swap memory. Web server can be thought that there are some processes that wait for I/O or

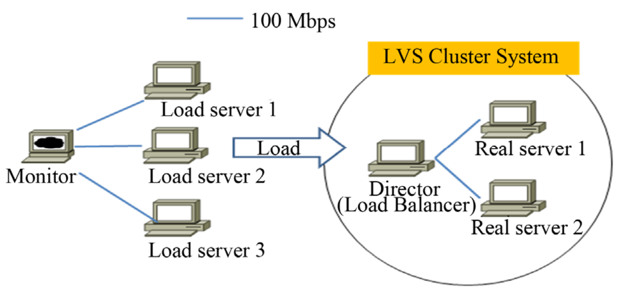

Figure 3. LVS clustering system.

other events, if the uninterruptible process value is non-zero. This is because if this value keeps being non-zero, one can judge there is a throughput problem. Finally, if the number of swapped out process is non-zero, the system shows lack of the amount of physical memory. If there is enough memory in page-in and page-out processes, only page-in and page-out will be occurred, but otherwise, unnecessary processes are swapped-out from the present memory, and the data of previously swapped out processes are swapped-in from the disk. In this process, CPU time is dissipated relatively to do swapping, so throughput of system is remarkably decreased. If the number of swapped out processes is steadily non-zero, it means a lack of memory, so it can affect the capability of the system [9]. To estimate load for each real server, the above mentioned parameters are measured by using vmstat, which is one type of UNIX.

In this research, there are some assumptions. The first assumption is that the capability of CPU is not considered in the view of distributing the amount of available web server memory equally. This is because the capability of web server is more affected by the amount of available memory than the capability of CPU. The ability of director is not considered is the second assumption. That is, even if there are many clients' requests, the series processes that make the requests head to real servers again do not cause huge load.

4.2. Load Generating Procedures

In this study, after virtual clients produce requests to transfer to LVS cluster system, they transfer to LVS cluster system to make diverse connections virtually headed for LVS cluster system. 100 virtual clients are produced every 30 seconds for 15 minutes in a load server. WebLoad is used for the tool to produce virtual clients in this research, and the procedures of WebLoad are as follows.

- Actual client visits a specific site, and write out an agenda file that contains the contents to request using script text.
- Produce many virtual clients that repeat contents of the composed Agenda file for designated time, and connect to LVS cluster system.
- Request the selected contents of Agenda file, and add a load to the specific site. If the request is done, and the results are returned, make a record, and virtual clients will be disappeared.

The Agenda file that virtual clients have is composed of the object site's html documents, graphic files, and a 1 kbyte message at CGI bulletin board, and 40 contents including the object site's html documents, graphic files, and CGI applications are required in this research. When a virtual client connects to the object site, 3.6 MByte memory is consumed due to HTTPD process, and after that, if the client requests CGI bulletin board, a son process of 1.7 MByte memory is produced. However, it occupies swap space as well as physical memory space.

Normal html documents, gif formed icon or banner, and database are not used for The web site contents that are not used in the network simulation of this research, and the simulation consists of CGI bulletin board that are programmed by Perl, which is a Interpreter method that process file-level I/O. Each real server's estimated results of LVS cluster system are saved in the database of director. There are 3 tables at the database in **Figure 4**, and the attributes in each table have identical names, but the same attributes are in two tables.

vmstat_1 and vmstat_2 are composed of not having any relationship to examine time difference of the estimated capability results for each server. It can be said that the capability estimation for each real server during the research is measured almost simultaneously, because the time difference to compose between the data of vmstat_1 and vmstat_2 is about 1~2 seconds.

vmstat_1 and vmstat_2 are the tables that save the capability results of real server1 and real server2, and the attributes are as follows.

id: the order that each data is stored (auto increment);

date: time when the data, that is, r, b, w, free, idle values are stored;

r: the number of process that are produced now, and running;

b: the number of uninterruptible process;

w: the number of swapped out process;

so: the amount of data that are swapped out for a second (KByte/s);

idle: the percentage of pause time for the entire CPU.

5. Analysis of Network Simulation

In the LVS cluster system by WLC algorithm, resource loss of real server is not considered and a manager selects the weights at random. LVS cluster system assigns the following connection with the server having value (1) as weight and the number of connections per each real server. **Figure 5** indicates the amount of free memory of two real servers that compose LVS cluster system, when

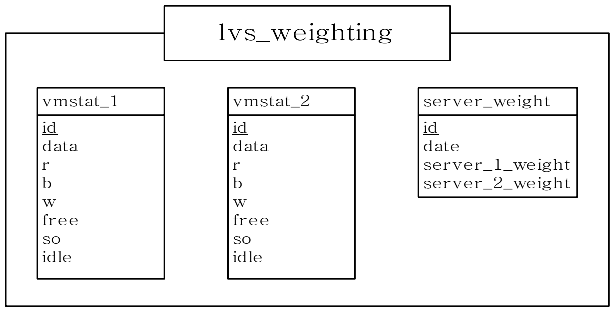

Figure 4. Attributes in database.

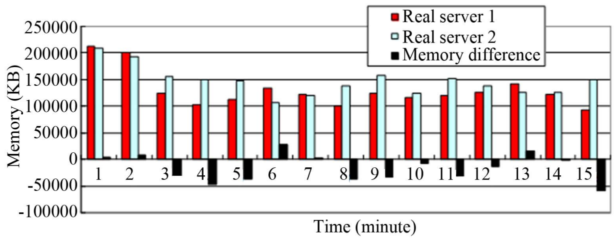

Figure 5. Amount of free memory under WLC algorithm.

constant load is given to LVS cluster system. In the experiment, when approximately 4000 requests for every 30 seconds are connected to LVS cluster system, a change of memory for each real server is indicated. Each real server is booted simultaneously, and the amount of available memory for each server is the same. The default weights of each real server are 1.

From the result of the experiment, one can see that more loads were assigned to real server 2. Requests with returned results having small magnitude are assigned to real server 1, and requests for real server 2 have the results with relatively large magnitude, for instance there is CGI. This is because CGI requests, transferred to the server with small connection, require much more memory than all the memory for html, or small image files transferred so far So it has equal variance when it comes to the number of connections, but it's not possible to forecast that it has equal variance in the aspect of the assumption of memory resources. If this process is repeated by a raft of clients, memory consumption can place too much on a specific server. Throughout all experiments, agenda files that virtual clients perform are the same Agenda, but requests have their own magnitude of returned results, which is shown in **Figure 6**.

In the meantime, another simulation results by alternative 1 and 2 is shown in terms of memory consumption and change of its weight in **Figures 7** and **8**, respectively. Network simulation is performed under LVS cluster system circumstance and condition that load effect is not considered, and fault weight is 1 for both real server 1 and 2. Also, a series of processes that modulate each server's weight every minute in director are included. The simulation results are that memory consumptions of real server 1 and real server 2 were relatively balanced mostly, even though memory consumption of each real server was placed much on one particular server partly. **Figures 7** and **8** show memory consumption and its corresponding weight difference, which is not tend to be balanced between times, it is because uninterruptible process was caused after 3 minutes of the beginning of the experiment. That means fault requests are rushed from clients and bottleneck happened due to the disk of real server 2, which is much slower than rapid CPU or memory as much as 1000 times in the process reading data from disk, so weight became 0. If weight becomes 0,

```
do GET("http://210.107.211.59/책표지/learning_perl.gif") {
}
do GET("http://210.107.211.59/cgi-bin/wowboard/board.cgi") {
}
do GET("http://210.107.211.59/image/write.gif") {
}
do GET("http://210.107.211.59/image/fix0.gif") {
}
do GET("http://210.107.211.59/image/delete0.gif") {
}
do GET("http://210.107.211.59/image/list0.gif") {
}
do GET("http://210.107.211.59/image/back0.gif") {
}
do GET("http://210.107.211.59/image/next0.gif") {
}
do GET("http://210.107.211.59/image/admin.gif") {
}
do
GET("http://210.107.211.59/cgi-bin/wowboard/board.cgi?bd=&list=0&j=form"
) {
}
do POST("http://210.107.211.59/cgi-bin/wowboard/board.cgi") {
    FORMDATA {
        j = "write";
        u = "";
        bd = "";
        id = "kalsman";
        passwd = "kalsman";
        email = "kalsman@keobuksun.kmu.ac.kr";
        upfile ="";
        title = "2010"
        content = "tested in 2010" ;
    }
}
```

Figure 6. Agenda for network simulation.

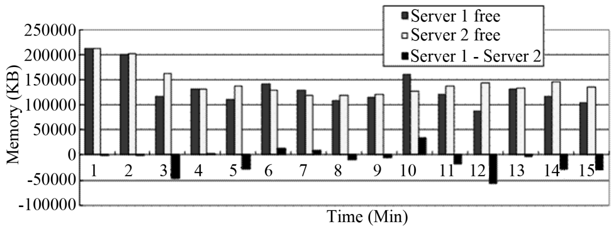

Figure 7. Amount of free memory under alternative 1.

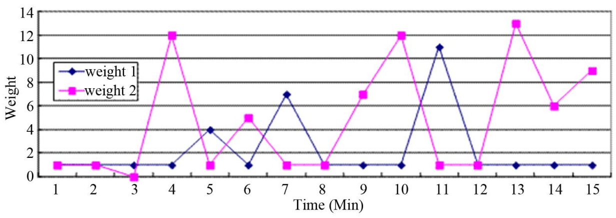

Figure 8. Weight difference under alternative 1.

director of LVS cluster system do not assign clients' requests to real server 2.

As a result, after 3 minutes of the experiment, the memory of real server 1 consumed a lot as seen in **Figure 7**. After that time, idle time percentage of CPU became 0, so CPU was not included to weight computation anymore. The reason why the results are generated might be because more html are required than CGI at the last part of the experiment.

All the simple process of alternative 2 is the same as those of method 1, and sensitivity of weight was tried to be desensitized applying (3) as a weight computation method. The results of a change for the amount of available memory of real server and weight difference are obtained by load distributing alternative 2 in **Figures 9** and **10**. As can be seen in **Figure 10**, the weight difference is changed relatively. For example, calculated weights after the first minute are 405 and 400 for real server 1 and real server 2, respectively, and 8 minutes later, 667 and 670. Real server has larger change of weight for the first few minutes, but gradually, weight of real server 2 and real server 1 intersect each other, which is called waveform. This phenomenon is because each real server's memory is consumed by turns so that memory consumption can be distributed equally. In the actual data, memory difference value was smaller than previous experiments, and it means correction potentiality was improved.

As can be seen in **Table 1**, it is clear that this method distributes memory consumption of each real server better than load distributing method 1, and mean of memory difference is decreased by half. Though simulation results were more improved than by WLC algorithm, alternative 1 does not meet our expectation. Because the improvements of LVS cluster system are caused by WLC algorithm. (1) is used to select a real server which will assign next connection in WLC algorithm. The number of remained connections in hash table of each real server is calculated by the sum of the number of active connections and inactive connections. **Table 1** indicates the results that the number of active connections and inactive

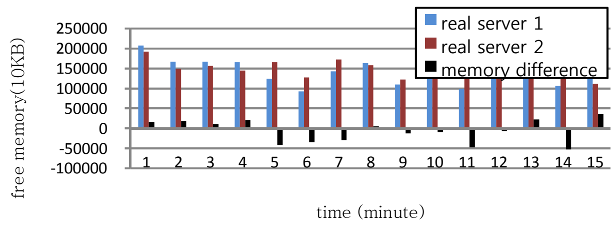

Figure 9. Amount of free memory under alternative 2.

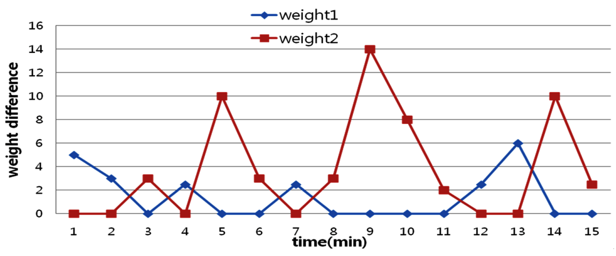

Figure 10. Weight difference under alternative 2.

Table 1. Average free memory (MByte).

	real server 1	real server 2	Memory difference
WLC algorithm	129680.1	145873.9	-16193.1
Alternative 1	132279.8	143608.1	-11328.3
Alternative 2	134530.9	141352.6	-6821.7

connections are measured at random. And at some point of time, when there are 4000 connections for every 30 seconds, servers are connected to cluster system for whole experiment.

LVS cluster system decides recorded connection to real server using the number of active connections and inactive connections. These connections affect actual real server memory the most, and connection numbers include the number of inactive connections and active connections in the process to calculate the weight. When a change of weight occurs at some point of time, many connections will be assigned to real server 1 if weight of real server 1 is larger than that of real server 2, and as a result, memory for real server 1 will be consumed more, and the number of the server connections will increase. Also, most of the connections will be inactive connections constantly. If this situation is repeated, a specific real server can have large weight and large numbers of inactive connections.

During a whole simulation, various experiments were performed, but it was not possible to correct unbalance of memory consumption perfectly. However, when a load is considered like alternative 2, resources of each real server were used effectively. The most important factor is to compute mean connection and weight per a minute for clients who use LVS cluster system. To compute the number of connections and weight, "access.log" file of each server needs to be analyzed and it has to be concluded by the number of HTTP process related to web service, the amount of memory that HTTPD process occupies, the amount of available memory at the measuring point of time, and process information on CGI or server API which is used on web site using tools like ps, vmstat, and uptime of UNIX.

6. Conclusions

The objective of this study is to suggest methods to utilize the information on the amount of load of real server to distribute server resources so that LVS cluster system which reduces memory unbalance phenomenon of servers can be suggested. The suggested method is to measure each real server's load periodically and from the measured results, compute weight of each real server in LVS cluster system, and apply the weight periodically. As a result, it was shown that load of real server could be reflected, and the phenomenon that a specific real server's memory of LVS cluster system is consumed constantly was complemented. Through actual experiments, it was proven that the suggested method in this research can treat system resources more effectively than a method that doesn't consider status information of real server loads.

The result of this research is a methodology that composes cluster system, and suggests a methodology that can compose cluster system with heterogeneous server beyond fixed ideas of IT society that server's specification has to be composed of same products or server with same capacity. That is, if server resources are properly distributed, effective cluster system can be composed. Though it is not mentioned in this research, the bandwidth of network, arrangement and magnitude of contents that compose web site, proper choice for CGI application language and application problems of database, and limit of service need to be considered as well. Collecting data from each real server itself can have load temporarily, because the accuracy of collected data is low due to load of real server itself.

REFERENCES

[1] D. A. Menasce and V. Almeida, "Capacity Planning for Web Performance," Prentice Hall PTR, 1998.

[2] K. Q. Li, "Minimizing the Probability of Load Imbalance in Heterogeneous Distributed Computer System," *Mathematical and Computer Modeling*, Vol. 36, 2002, pp. 1075-1084.

[3] S. Beak, H. Rim and S. Kim, "Socket-Based RR Scheduling Scheme for Tightly Coupled Clusters Providing Single-Name Image," *Journal of System Architecture*, Vol. 50, 2004, pp. 299-308.

[4] D. C. Li, C. Wu and F. Chang, "Determination of the Parameters in the Dynamic Weighted Round-Robin Method for Work Load Balancing," *Computers & Operations Research*, Vol. 32, 2005, pp. 2129-2145.

[5] E. M. Choi, "Performance Test and Analysis for an Adaptive Load Balancing Mechanism on Distributed Server Cluster Systems," *Future Generation Computer Systems*, Vol. 20, 2004, pp. 237-247.

[6] M. Hamdi and C. K. Lee, "Dynamic Load-Balancing of Image Processing Applications on Clusters of Workstations," *Parallel Computing*, Vol. 22, 1997, pp. 1477-1492,

[7] J. Mack and W. Zhang, "The Linux Virtual Server HOWTO," 1999. http://www.linuxvirtualserver.org/Joseph.Mack/LVS-HOWTO-991205.gz

[8] W. Zhang, S. Jin and Q. Wu, "Creating Linux Virtual servers," *Ottawa Linux Symposium*, 2000.

[9] J. Purcell, "Linux Complete Command Reference," Redhat Press, 1997.