Asian Journal of Information Technology 5 (11): 1284-1290, 2006

Capacity Based Load Balancing Scheme for Fair Request Dispatching

Tsang-Long Pao and Jian-Bo Chen
Department of Computer Science and Engineering, Tatung University Taipei, Taiwan, R.O.C

Abstract: The load balancing scheme can achieve high performance for the web server. Most of the load balancing architectures are based on homogeneous web servers. If the backend servers in the system are different hardware specifications, the load balancing system must have a strategy to evenly dispatch the load to the backend servers. In this study, we derive a formula to define the capacities for heterogeneous web servers. In the experimental results, the maximum connection number with certain drop rate can be the capacities, but it can not get fair response time for each client requests. We must consider the capacities not only depending on drop rate but also on the response time. Under this definition, the response time for all client requests will nearly be the same.

Key words: Load balance, heterogeneous web server, capacity

INTRODUCTION

Because of the flourishing development of the internet applications, the services offered by popular web sites become diversified. Therefore, using one single server to serve all the requests will be overloaded. Under such situation, we can use distributed or parallel architecture to make the website more efficient. The load balanced web server architecture can provide high performance services for large number of clients[1,2]. Although load balanced servers consist of one or more servers, they act as a single unit. Load balanced servers offer the advantages of user transparency that allows the clients to work with multiple servers without any specific configuration[3].

There are many solutions for solving the load balancing scheme, include client-side approach, DNS-based approach, server-side approach and dispatcher-based approach[4]. However, most of the load balanced algorithms are focused on the homogeneous web servers, that is, all of the web servers are the same specification. These web servers have the same processing powers, memory installation, network speed, I/O speed, etc[5]. In the homogeneous web server system, the dispatcher can redirect the client request to the most appropriate server based on many well-known criteria, such as Round Robin or Least Connection. In the server-state based algorithm[6], each web server must report its loading information to the dispatcher. The dispatcher then selects the best server to serve that request[7,8]. But in the heterogeneous web server system, the dispatcher should redirect the client request to the most appropriate server not only by considering the loading information but also by the capacity of the server. In other words, we use the capacity of the web server to decide which server is the most appropriate.

In order to avoid the user perceive latency on the web server, the capacity of heterogeneous web server must be well defined that every client request will achieve the fair response time. As we known, the more powerful backend server with higher capacity should serve more requests than others, but the response time maybe still shorter than others. In this situation, the client request which was redirected to the poor backend server might get longer response time. In this study, the backend server always busy in responding the client requests, so the backend server should avoid reporting their load information to dispatcher which will increase the load of the server. In this study, we will derive a formula to define the capacity which is independent of the server loading and every client requests will get nearly the same response time no matter what backend server it was redirected to.

Related works

Dispatcher-based approach: To centralize request scheduling and completely control client-request routing, a network component of the web server system acts as a dispatcher. Request routing among servers is transparent. The dispatcher uniquely identifies each backend server in the system through a private address that can be at different protocol levels, depending on the architecture.

Corresponding Author: Tsang-Long Pao, Department of Computer Science and Engineering, Tatung University Taipei, Taiwan, R.O.C

We differentiate dispatcher-based architectures by routing mechanism-packet single-rewriting, packet double-rewriting, HTTP redirection, or server-based HTTP redirection[4].

Packet single-rewriting: In some architectures, the dispatcher reroutes client-to-server packets by rewriting their IP address, such as in the basic TCP router mechanism. The web server cluster consists of a group of backend servers and a dispatcher that acts as an IP address translation. Figure 1 outlines the mechanism, in which address i is the IP address of the i-th web server.

All HTTP client requests reach the dispatcher because the IP-SVA is the only public address. The dispatcher selects a backend server for each HTTP request through a Round Robin algorithm and achieves routing by rewriting each incoming packet's destination IP address. The dispatcher replaces its IP-SVA with the selected server's IP address. Because a request consists of several IP packets, the dispatcher tracks the source IP address for every established TCP connection in an address table. The dispatcher can thereby always route packets regarding the same connection to the same web server. Furthermore, the web server must replace its IP address with the dispatcher's IP-SVA before sending the response packets to the client. Therefore, the client is not aware that its requests are handled by a hidden web server.

Packet double-rewriting: This mechanism also relies on a centralized dispatcher to schedule and control client requests but differs from packet single-rewriting in the source address modification of all packets between server and client. Packet double-rewriting is based on the Internet Engineering Task Force's Network Address Translator mechanism, as shown in Fig. 2. The dispatcher receives a client request, selects the web server and modifies the IP header of each incoming packet and also modifies the outgoing packets that compose the requested document.

HTTP redirection: A centralized dispatcher receives all incoming requests and distributes them among the web server nodes through the HTTP's redirection mechanism. The dispatcher redirects a request by specifying the appropriate status code in the response, indicating in its header the server address where the client can get the desired document[9]. Such redirection is largely transparent; at most, users might notice an increased response time. Unlike most dispatcher-based solutions, HTTP redirection does not require IP address modification of packets reaching or leaving the web server system. The HTTP redirection scheme is shown in Fig. 3.

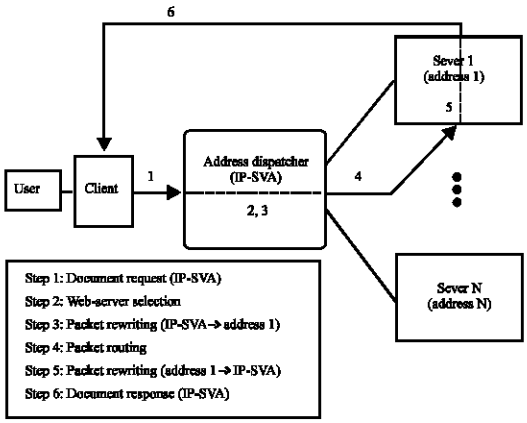

Fig. 1: Packet single-rewriting by the dispatcher

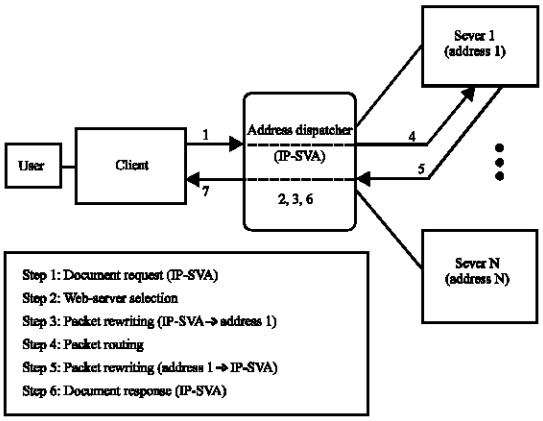

Fig. 2: Packet double-rewriting by the dispatcher

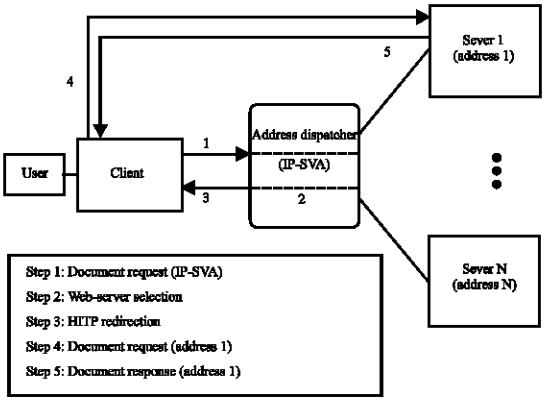

Fig. 3: HTTP redirection

Load balancing algorithms: The load balancing system makes decisions on which backend server should be assigned a new connection based on the load balancing algorithms. We will discuss various scheduling algorithms for selecting backend servers from the

cluster for new connections: Round-Robin, Weighted Round-Robin, Hash, Bandwidth, Least-Connection, Weighted Least-Connection and Respond Time[6,10]. The first four algorithms are self-explanatory, because they don't have any load information about the servers. The last three algorithms count active connection number or detect the situation for each server then estimate servers load based on those connection numbers and server response time.

Round-robin scheduling: Round-Robin scheduling algorithm directs the network connections to the different backend servers in the Round Robin manner. It treats all backend servers as equals regardless of number of connections or response time. That is, the first backend server in the group gets the first connection; the second backend server gets the next connection, followed by the third backend server and so on. When all the backend servers in this group have received at least one connection, the process starts over with the first backend server.

Weighted round robin scheduling: The Weighted Round-Robin(WRR) scheduling can treat the backend servers of different processing capacities. Each server can be assigned a weight, an integer that indicates its processing capacity, the default weight is 1. In the WRR scheduling, all servers with higher weights receives new connections first and get more connections than servers with lower weights, servers with equal weights get an equal distribution of new connections. For example, the backend servers A, B, C have the weights 2, 3, 4 respectively, a good scheduling sequence can be CCBCBACBA in a scheduling period (mod sum(Wi)). The WRR is efficient to schedule request, but it may still lead to dynamic load imbalance among the real servers if the load of requests vary highly.

Hash scheduling: When selecting a backend server, a mathematical hash of the relevant IP address information is used as an index into the list of currently available servers. Any given IP address information will always have the same hash result, providing natural persistence, as long as the backend server list is stable. However, if a server is added to or left the system, then a different backend server might be assigned to a subsequent session with the same IP address information even though the original server is still available. Open connections are not cleared.

Bandwidth scheduling: The bandwidth algorithm uses backend server octet counts to assign sessions to a server. The dispatcher monitors the number of octets sent between the server and itself. Then, the backend server weights are adjusted so they are inversely proportional to the number of octets that the backend server processes during the last interval.

Backend servers that process more octets are considered to have less available bandwidth than those that have processed fewer octets. For example, the backend server that processes half the amount of octets over the last interval receives twice the weight of the other backend servers. The higher the bandwidth used, the smaller the weight assigned to the server. Based on this weight, the subsequent requests go to the backend server with the highest amount of free bandwidth. These weights are automatically assigned.

Least-connection scheduling: The least-connection scheduling algorithm directs network connections to the server with the least number of active connections. This is one of dynamic scheduling algorithms, because it needs to count active connections for each backend server dynamically. The backend server with the fewest current connections is considered to be the best choice for the next client connection request. This algorithm is the most self-regulating, with the fastest servers typically getting the most connections over time. At a virtual server where there is a collection of servers with similar performance, the least connection scheduling is good to smooth distribution when the load of requests vary a lot, because all long requests will not be directed to a single server. At a first look, the least-connection scheduling can also perform well even if servers are of various processing capacities, because the faster server will get more network connections. In fact, it cannot perform very well because of the TCP's TIME-WAIT state. The TCP's TIME-WAIT is usually 2 minutes, in which a busy web site often gets thousands of connections. For example, the server A is twice as powerful as the server B, the server A has processed thousands of requests and kept them in the TCP's TIME-WAIT state, but the server B is slow to get its thousands of connections finished and still receives new connections. Thus, the least-connection scheduling cannot get load well balanced among servers with various processing capacities.

Weighted least-connection scheduling: The Weighted Least-Connection s cheduling is a superset of the least-connection scheduling, in which a performance weight can be assigned to each server. The backend servers with a higher weight will receive a larger percentage of active connections at any time. The virtual server administrator can assign a weight to each backend

server and network connections are scheduled to each server in which the percentage of the current number of active connections for each server is a ratio to its weight. The Weighted Least-Connections scheduling works as follows: supposing there is n backend servers, each server I has weight Wi (i=l,…,n) and active connections Ci (I=l,…,n), all connection number S is the sum of Ci (i=l ,..,n), the network connection will be directed to the server j, in which (Cj/S)/Wj= min{(Ci/S)/Wi}(i =l,...,n) Since the S is a constant in this lookup, there is no need to divide Ci by S, it can be optimized as Cj/Wj = min{Ci/Wi} (i=l,...,n). Since there is no floats in Linux kernel mode, the comparison of Cj/Wj > Ci/Wi is changed to Cj * Wi > Ci * Wj because all weights are larger than zero.

Response time scheduling: The response time algorithm uses backend server response time to assign sessions to servers. The response time between the servers and the dispatcher is used as the weighting factor. The dispatcher monitors and records the amount of time it takes for each backend server to reply to a health check to adjust the backend server weights. The weights are adjusted so they are inversely proportional to a moving average of response time. In such a scenario, a server with half the response time as another server will receive a weight twice as large.

System architecture

Server selection algorithm: The proposed load balancing architecture consists of a selector, a center database server and a number of heterogeneous web servers. In order to keep track of the flow between servers and clients, there must have a log server in our architecture. The log server is used to check whether the requests are distributed evenly and we can decide to add new backend server if needed. In other words, we must make an effort to achieve the maximum performance with existing servers by more sophisticated load balancing algorithm before adding extra backend servers.

The operations of the system are described as follows. First, web servers register to the database. The database server will generate a serverlist table on the database. When client issues request to selector, the selector looks up the serverlist table to achieve the most appropriate server's IP address. Then selector returns an HTML document with HTTP header redirection to indicate the backend server's IP address to the client. After client receives the document, it issues the request to the appropriate web server. The appropriate server serves the client requests until connection finished. Figure 4 shows the processes of server selection algorithm.

Fig. 4: Server selection algorithm

The definition of capacity: Assume that there are 3 heterogeneous web servers in the load balancing system, say S_1,S_2,S_3. We try to define the capacities of these servers C_1,C_2,C_3. The capacities are the weights of these web servers. And using the weights in Weight Distributing Algorithm, we can achieve high performance and fair client response time.

Capacity measurement: First of all, we must know the maximum connection of a single server. When defining the maximum connection of a server, we must define a threshold that the drop rate is below that value. Assume that the clients issue M requests to the server, the server will response the request in a certain time individually. We consider that the response time R have p distinct values: $0<R_1<R_2<\ldots<R_p<\infty$. We partition the possible values into n disjoint intervals, $\{[T_0=0,T_1), [T_1,T_2),\ldots, [T_{n-1},T_n=\infty)\}$, where each R will fall in the ith interval $[T_{i-1},T_i)$ and the connection number for $[T_{i-1},T_i)$ is m_i, where $m_1+m_2+\ldots+m_n=M$. For user perceive latency, we define the maximum response time R_{max} as the threshold which is fall in the jth interval $[T_{j-1},T_j)$. For each request, if the response time is higher than the R_{max}, then we consider that the request was dropped. We define the drop rate when the R_{max} fall in the jth interval as shown in Eq. 1. If the drop rate is under certain percentage, we can obtain the maximum connection or the capacity for that server in Eq. 2.

$$\text{Drop rate}: D = \frac{\sum_{j}^{n} m_i}{M}, \text{where } R_{max} \text{ fall in the jth interval} \quad (1)$$

$$\text{Capacity}: C = \sum_{1}^{j-1} m_i, \text{where } R_{max} \text{ fall in the jth interval} \quad (2)$$

for each time intervals $\{[T_0=0,T_1), [T_1,T_2),\ldots, [T_{n-1},T_n=\infty)\}$
  where the connection number for $[T_{i-1},T_i)$ is m_i
  and $m_1+m_2+\ldots+m_n=M$
the maximum threshold R_{max} is fall in the jth interval $[T_{j-1},T_j)$
  Drop rate $D=(m_j+m_{j+1}+\ldots+m_n)/M$
the minimum threshold R_{min} is fall in the kth interval $[T_{k-1},T_k)$, where $k<j$
  Capacity $C=m_1+m_2+\ldots+m_{k-1}$

Fig. 5: The algorithm for drop rate and capacity

For example, if the total request number is 500 (M=500). The response time have 10 distinct intervals {[0,0.25),[0.25,0.5),[0.5,0.75),[0.75,1),[1,1.25),[1.25,1.5),[1.5 ,1.75), [1.75,2), [2,2.25), [2.25,∞)} and the connection member for each interval is {120,95,75,65,45,30,25,22,15,8}. If we define the threshold as 2 second (R_{max}=2), then the drop rate D=(15+8)/500=4.6% and the capacity is C=(120+95+75+65+45+30+25+22)=477.

Capacity enhancement for fair response time: We find that the more powerful server S_3 will serve more client requests because we define higher capacity for that server. But in the figure of response time analysis, the S_3, which serves more client requests, still can response faster than others. It is not fair for user perceive latency. We need to consider both the drop rate and response time in the process of capacity decision making. Using the same environment as above, the drop rate definition is still the same, but we want to modify the definition of the capacity. Rather than only consider the drop rate, we use the response time as another parameter to calculate the capacity. Assume that we define the acceptable response time R_{min} as the threshold, that means the capacity definition must under the minimum threshold rather than the maximum threshold R_{max}. If the R_{min} is fall in the kth interval $[T_{k-1},T_k)$, where $k < j$. We can re-define the capacity in Eq. 3.

$$\text{Capacity}: C = \sum_{1}^{k-1} m_i, \text{where } R_{min} \text{ fall in the kth interval} \quad (3)$$

For the above example, we define the threshold as 1 second (R_{min}=1), then the capacity is C=(120+95+75+65)=355. Under this definition, the experimental results in shows that the client request will almost got the same response time. The algorithm for the drop rate and capacity are given in Fig. 5.

Weighted distributing algorithm: In this study, we will show the distributing algorithm we used in the server selection process. In our load balancing scheme, each backend server has different capacities. The capacities are the weights of the server. After obtaining the capacity for each server, the serverlist table is generated. Figure 6 shows an example.

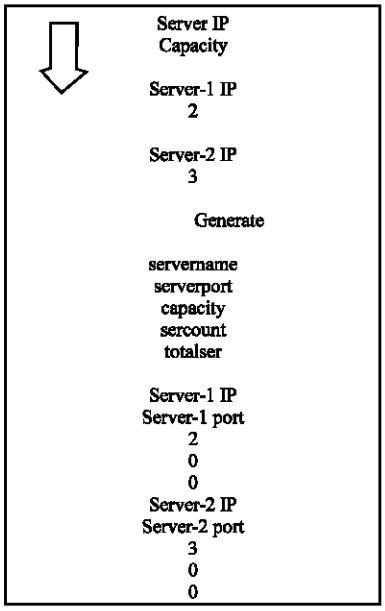

Fig. 6: Example of serverlist table

The IP address of individual web server is stored in servername field. The serverport field specifies the TCP port number of individual web server. The sercount field stores the correct connections and the totalser field stores the total counts of each web server.

When a client issues a request to the selector, the selector communicates with the database and retrieves the serverlist table. The selector will run an algorithm by calculating the value of subtracting sercount from capacity on each entry and then the selector will choose the biggest one and return the server's IP to client. If the values are same, the selector will return the server's IP which ranks at the more preceding one. And then add sercount by one at the selected entry. Until the value of capacity is equal to sercount on each entry, then the selector add sercount to totalser and reset sercount to zero.

RESULTS AND DISCUSSION

To verify our proposed architecture can be applied in the heterogeneous system, we use three servers with different computational power as the backend servers. The servers are named S_1, S_2 and S_3. Table 1 shows the serverlist Table and Fig. 7 shows the maximum connection numbers per second of each web server.

Figure 8 and 9 show the drop rate and the average response time about these three web servers with different abilities. On Fig. 7 to 9, we can see that the high-end web server can handle more client requests at the same time. Furthermore, the high-end web server takes

Table 1: Serverlist table

Server name	CPU
S_1	1.0G Hz
S_2	1.6G Hz
S_3	2.6G Hz

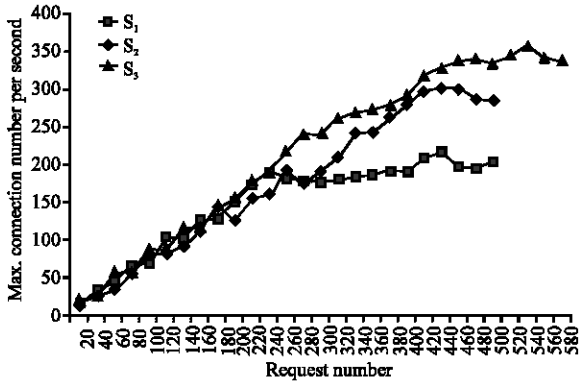

Fig. 7: Maximum connection per second

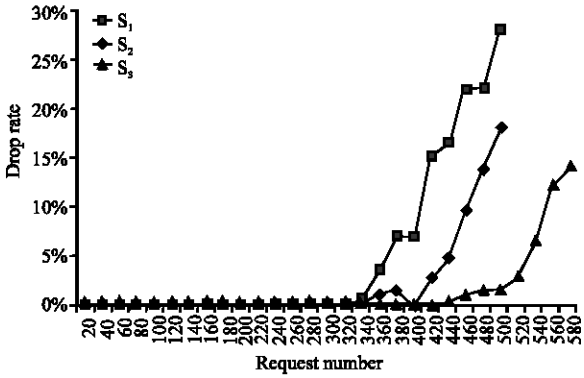

Fig. 8: Drop rate

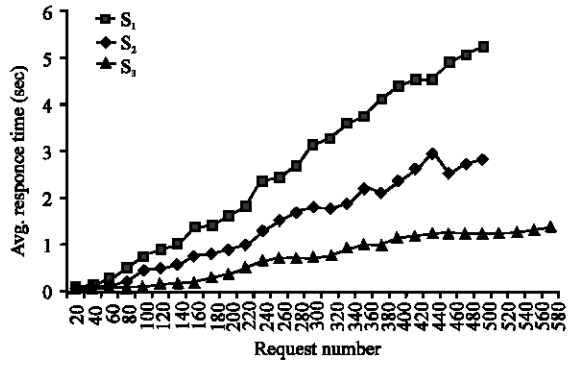

Fig. 9: Average response time

little time to response the same number of clients' requests. Based on the experimental results above, we got some conclusions on Table 2.

The next experiment is to set the capacity value for the heterogeneous web servers. The capacity values

Table 2: Comparison of three web servers

Server	CPU	Max. request numbers - Drop rate is below 5%	Max. request numbers - Drop rate is below 5% - Average response time is below 1 second
S_1	1 GHz	360	120
S_2	1.6 GHz	440	200
S_3	2.6 GHz	520	340

Table 3: Capacity of each web server-drop rate below 5%

Server	Max. request numbers - Drop rate is below 5%	Capacity
S_1	360	9
S_2	440	11
S_3	520	13

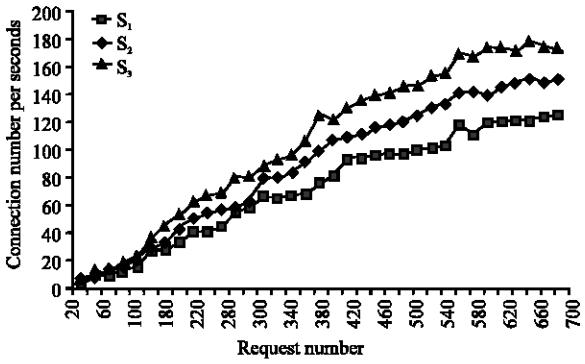

Fig. 10: Connection numbers per second-capacity ratio is 9:11:13

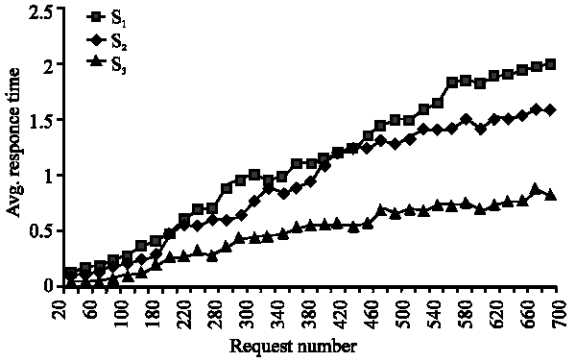

Fig. 11: Average response time-capacity ratio is 9:11:13

derived from the column of maximum request numbers under 5% drop rate in Table 2 shows in Table 3.

Figure 10 and 11 show the experimental results when the capacity ratio of S_1:S_2:S_3 is 9:11:13. The highest capacity server S_3 can serve more request than the others in Fig. 10. Although S_3 can server more requests, the average response time is still shorter than S_1 and S_2. In fact, it is unfair for users to connect to our load balancing system under the rule of first come first serve. The reason is that we define the capacity value only considering the drop rate.

We consider both drop rate and average response time to get new capacity ratio. The new capacity ratio of

Table 4: Capacity of each web server-drop rate below 5% and avg. response time below 1 second

Server	Max. request numbers - Drop rate is below 5% - Avg. response time is below 1 second	Capacity
S_1	120	6
S_2	200	10
S_3	340	17

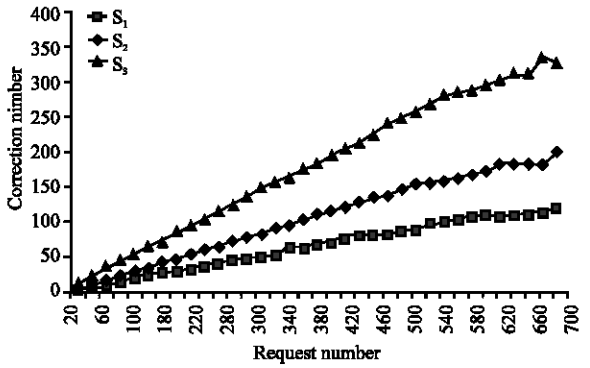

Fig. 12: Connection number per second-capacity ratio is 6:10:17

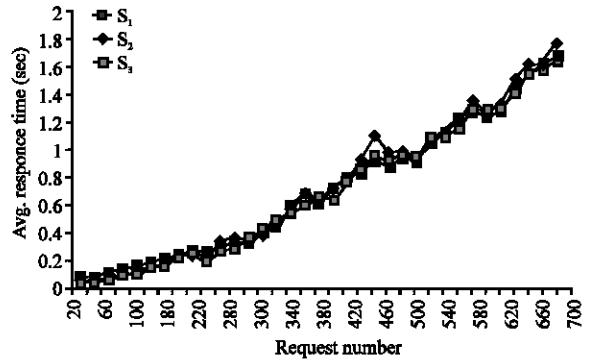

Fig. 13: Average response time-capacity ratio is 6:10:17

S_1:S_2:S_3 is 6:10:17 shown in Table 4 which is derived from the column of average response time under 1 second in Table 2. Figure 12 and 13 show the connection number and average response time of these web servers. The values of average response time for the three servers are very close.

CONCLUSION

In order to implement the load balance architecture of heterogeneous web servers, there must be an appropriate distributing rule. For the objective of servicing users fairly, each request from users should have the same response time. This study derived a formula to define the capacity so that each heterogeneous web server can serve the client in a certain response time. We adjust the capacity of each web server and prove that the method we proposed is useful. The experimental results also show that the system can distribute the requests to the servers according to their capacities.

REFENENCES

1. Mosedale, D., W. Foss and R. McCool, 1997. Lessons Learned Administering Netscape's Internet Site, IEEE Internet Computing.
2. Katz, E.D., B. Michelle and Robert McGrath, 1994. A Scalable HTTP Server: The NCSA Protocol, Proc. First International Conference on the World-Wide Web.
3. Cardellini, V., C. Michele and S.P. Yu, 2003. Request redirection algorithms for distributed Web systems, IEEE Trans. Parallel and Distributed Systems.
4. Cardellini, V., C. Michele and S.P. Yu, 1999. Dynamic load balancing on web-server system, IEEE Internet Computing, pp: 28-39.
5. Michele Colajanni, S.P. Yu and D.M. Dias, 1998. Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems, IEEE Trans. Parallel and Distributed Systems, pp: 585-600.
6. Tsang-Long Pao, Jian-Bo Chen and I-Ching Cheng, 2004. An Analysis of Server Load Balance Algorithms for Server Switching, Proc. Ming-Chung University International Academic Conference.
7. Xin Liu and A. Andrew Chien, 2003. Traffic-based load balance for scalable network emulation, Proc. ACM/IEEE Supercomputing.
8. Schroeder, T., S. Goddard and B. Ramamurthy, 2000. Scalable Web server clustering technologies, IEEE Network.
9. W3C World Wide Web Consortium, http://www.w3c.org.
10. Wensong Zhang, Shiyao Jin and Quanyuan Wu, 2000. Scaling Internet Services by Linux Director, Proc. High Performance Computing in the Asia-Pacific Region.