WWW Server Load Balancing Technique Employing Passive Measurement of Server Performance

Satoru Ohta[1] and **Ryuichi Andou**[2], Non-members

ABSTRACT

Server load balancing is indispensable within World Wide Web (WWW) for providing high-quality service. In server load balancing, since the server loads and capacities are not always identical, traffic should be distributed by measuring server performance to improve the service quality. This study proposes a load balancing technique conducted by passive measurement, which estimates the server performance via user traffic passing through the load balancer. Since this method evaluates server performance without executing any programs on the server, no additional server or network load is generated. This paper first presents a server performance metric that can be passively measured. The presented metric utilizes the characteristics of TCP SYN and SYN ACK messages exchanged in the TCP connection establishment phase. An experiment shows that the metric correctly identifies server performance degradation. The paper then proposes a load balancing algorithm based on the metric, and its implementation issues. The proposed algorithm distributes fewer requests to servers that do not have sufficient capacities. Because of this, the algorithm achieves good performance in a heterogeneous environment where servers with different capacities coexist. The effectiveness of the proposed load balancing technique is confirmed experimentally.

Keywords: IP Networks, Load Balancing, WWW, TCP, Passive Measurement, Performance

1. INTRODUCTION

The World Wide Web (WWW) is the most important service provided by the Internet. The quality of the service depends on the capacity of the servers; if service demand exceeds the capacity of a server, multiple servers are employed to share the load, and achieve an acceptable service quality [1, 2].

The following characteristic must be considered in order to establish a powerful load balancing technique.

- The specifications of the servers are not necessarily identical; servers with different capacities may coexist.
- The server load incurred by a request is not constant. Some requests may offer much heavier load than others.

Because of these characteristics, the performance of some servers is relatively degraded. Thus, the server load may become imbalanced if requests are distributed to each server with equal probability.

The above problem is avoided by measuring the performance of each server and distributing fewer requests to servers that do not have sufficient capacities. Such control has been achieved by executing a load/performance measurement program on each machine and by transmitting the measurement result to the load balancer [1, 2]. The load balancer can then distribute requests to server considering their load/performance. However, by executing the measurement program on the servers, this method may generate additional processing loads on them. Moreover, if a servers performance is extremely degraded with excessive loads, the program may not function efficiently.

As an alternative method, this study investigates the passive measurement approach, in which server performance is estimated via the user traffic passing through the load balancer. Since this approach does not generate any additional loads on the servers or the network, the estimation is not affected by measurement program/packets. Passive measurement approaches have been reported in [3, 4]. However, the method proposed in the present study is more advantageous because its performance metric is obtainable by simpler computation. The Linux Virtual Server [5, 6] and NetDispatcher [7] are also load balancing methods based on passive measurement. These methods distribute load among servers by estimating the number of TCP connections. However, the number of TCP connections does not necessarily represent the server performance and thus is not an appropriate metric for load balancing in a heterogeneous environment. By contrast, the metric presented in this paper correctly identifies server performance. Because of this, the proposed method is more effective when some servers are more degraded than others.

The proposed measurement method uses the fact

Manuscript received on August 1, 2009 ; revised on November 5, 2009.

[1,2] The authors are with the Department of Information Systems Engineering, Faculty of Engineerig, Toyama Prefectural University, 5180 Kurokawa, Imizu-shi, Toyama, 939-0398, Japan., Email: ohta@pu-toyama.ac.jp

that server performance degradation is identified through the packets exchanged in the establishment phase of a TCP connection. Based on this idea, this paper presents a new server performance metric that can be measured passively, and subsequently applies the performance metric to load balancing. The paper describes the algorithm that controls the probabilities of distributing requests according to the performance metric. The effectiveness of the proposed metric and the load balancing technique is confirmed experimentally.

The proposed load-balancing technique is applicable to any TCP server-client based services. However, since WWW is the most heavily used service in the Internet, the load-balancing for WWW is more frequently required. Therefore, this paper focuses on WWW as the application of the proposed method.

The paper first overviews related studies and describes the problem tackled in this study in Sect. 2. Section 3 presents a server performance metric, estimated through passive measurement. This section shows that the metric is closely related to the TCP connection time. Section 4 details the proposed load balancing technique. Finally, Sect. 5 concludes the paper.

2. RELATED WORKS AND PROBLEM DESCRIPTION

So that the load balancer maximizes system performance, service requests should not be distributed to all servers with equal probability . This is because WWW services often involve complex processing, and the load induced by the processing varies greatly with the content of a request. As a result, the load on each server is not always the same even if every server receives the same number of service requests. Further, the capacities of servers are not always identical. For example, suppose that the service demand has increased, and the current number of servers is insufficient for handling requests. In this case, new server machines must be added to the system, and it is very likely that the capacities of the newly purchased machines are larger than those of the old machines. Moreover, the performance of a server may degrade comparatively because of faults such as RAID disk trouble.

To solve the above problem, we measure the performance of each server and distribute service requests based on this performance, reflecting the remaining capacity of the machine. More requests need to be distributed to a server with greater remaining capacity and fewer requests to a server with less remaining capacity. Therefore, advanced load balancing techniques employ a mechanism, to estimate the load/performance of servers.

Existing load balancing techniques often employ a program running on each server to measure its load/performance [2, 8]. Unfortunately, with this method, the program itself generates additional processing load on the server, deteriorating its performance. Moreover, if a server's performance is extremely degraded with excessive load, the measurement program may not work efficiently. Thus, any method based on a measurement program executed on the server is not reliable.

An alternative regarding server load/performance estimation is to measure the response time of the ICMP echo message. However, this response time does not always represent the load or performance of the server accurately. Moreover, the ICMP messages offer additional traffic to the network and the servers, affecting system performance.

As seen above, existing techniques have disadvantages relating to additional loads or inaccuracy. To establish an ideal load balancing method, the server load/performance must be estimated through passive measurement, monitoring only the user traffic. This approach does not insert any probe packets into the network or execute any programs on the server machines.

Load balancing techniques based on passive measurement have been examined in [3, 4]. For the performance metric, the method of [3] employs the application-layer round-trip time, while the method of [4] employs the flow rate. For the estimation of these metrics, the variables used for the computation must be managed individually for each TCP connection. Since a server handles many TCP connections, the estimation of the metrics requires a large amount of storage and computational time. Thus, it is necessary to establish a performance metric that is passively measured and estimated by a simpler process than the methods of [3, 4]. In addition, these methods have been evaluated only by computer simulations and have not been implemented on real networks. Thus, their feasibility is unclear.

Implemented load balancing techniques include the Linux Virtual Servers [5, 6] and NetDispacher [7]. The former uses the number of current TCP connections as the server load metric while the latter uses the number of TCP connections as well as server machine information gathered by agent programs. In these techniques, the number of TCP connection is passively measured. However, the number of TCP connections is not directly related to the server capacity or performance. Obviously, a high-capacity server machine can accept many concurrent connections while a low-capacity machine can accept fewer connections. Meanwhile, the techniques consider that a machine with fewer connections is lightly loaded. Thus, the techniques route a new connection to the machine that has the fewest connections. This operation minimizes the difference among the number of TCP connections handled by each machine. Therefore, the techniques will offer a similar number of TCP connections to a high-capacity machine as that

to a low-capacity machine. It would therefore appear that this operation is not adequate for the case in a heterogeneous environment. For load balancing in heterogeneous environments, it is necessary to develop a metric that reflects server performance and can be passively measured.

Several studies have indicated that the performance of a WWW server is closely related to the socket accept queue [9–11]. If excessive load is offered to a server, the socket accept queue buffer overflows with TCP SYN messages, and if the message is lost, the client retransmits the TCP SYN message after the timeout period. This timeout and retransmission process greatly increases the connection time. Thus, if the accept queue overflow is identified from the user traffic, excessive server load and performance degradation can be detected effectively.

By utilizing the above characteristics, Refs. [9, 10] examine server capacity estimation based on TCP SYN drop rate for the purpose of energy conservation. In [12], the ratio of SYN ACK messages and TCP SYN messages is used to detect SYN flood Denial of Services (DoS) attack. However, the abovementioned characteristics of TCP SYN and SYN ACK messages have not been investigated from the viewpoint of server load balancing.

The technique reported in Reference [13] utilizes TCP SYN/SYN ACK messages in a different way to distribute load among multiple access links. The technique passively measures round trip times by recording arrival times of TCP SYN and SYN ACK messages. Load is then distributed among access links according to the measured round trip times. To execute this method, the arrival times of TCP SYN and SYN ACK messages must be managed individually for each TCP connections. Therefore, if many TCP connections are set up in the network, the technique will require a large amount of processing load and storage. By contrast, the approach of this study does not need to manage each connection and thus is achieved with much less processing load and storage

A different aspect of load-balancing is found in Ref. [14], which reports on a content switch. The content switch distributes load according to the contents of service requests and not according to server performances. Therefore, the method of Ref. [14] is not necessarily effective for heterogeneous environment, which is the target of this study. From the viewpoint of TCP SYN/SYN ACK message utilization, the method of Ref. [14] finds a pre-allocate server from the source IP address shown in a TCP SYN packet to reduce the processing load of converting sequence numbers. The method also uses the sequence numbers indicated in TCP SYN/SYN ACK messages for converting the sequence numbers. To perform this, the IP addresses and the sequence numbers must be managed individually for each TCP connection. For this reason, the method of Ref. [14] consumes a large amount of storage and computational time. Meanwhile, the approach of this study manages only the numbers of TCP SYN and SYN ACK packets and does not manage each connection. Therefore, its computational procedure is much simpler than that of the method of Ref. [14].

The numbers of TCP SYN and SYN ACK messages can be measured passively and represent server performance degradation. This characteristic is well suited to load balancing in heterogeneous environments. The following sections describe how this is achieved.

3. SERVER PERFORMANCE METRIC

3.1 Theory

An important aspect of server performance is the connection time, which is defined as the period needed to set up a TCP connection. The connection time depends on several factors, such as packet communication delay in the network or processing delay at the server. However, it is reported that the most dominant factor affecting connection time is the buffer overflow of the socket accept queue [9, 11]. The retransmission process for a lost TCP SYN message takes a few seconds, while the other delays are shorter than one second. Thus, other delays are almost negligible in comparison with those caused by TCP SYN retransmission.

Let t_0 be the timeout period for the TCP SYN retransmission. We assume that the first TCP SYN is lost, and the retransmitted second TCP SYN establishes the TCP connection successfully. Thus, the connection time is greater than t_0. Similarly, the lower bound of the connection time is nt_0 for n retransmissions of a TCP SYN message. Moreover, the connection time can be approximated by nt_0 assuming that other delays are much smaller than t_0.

If a TCP SYN is accepted, the server sends back a SYN ACK message to the client. Thus, the number of retransmitted TCP SYN messages is estimated by counting the number of TCP SYN messages transmitted before receiving a SYN ACK message. If N_A TCP connections are established for a specified period, the server generates N_A SYN ACK messages. Let N_S be the number of TCP SYN messages sent to the server for that period. The number of overflow TCP SYN messages is then approximated by $N_S - N_A$. The average number of the retransmitted TCP SYN messages for a connection is thus $(N_S - N_A)/N_A$. This implies that the average connection time t_c is approximated by the following equation.

$$t_c \approx \frac{(N_S - N_A)t_0}{N_A}. \tag{1}$$

The above equation suggests that the following value R is an effective metric for server performance.

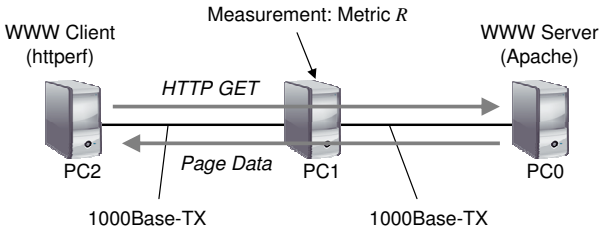

Fig.1: *Network configuration to evaluate the performance metric R.*

$$R = \frac{N_S - N_A}{N_A}. \tag{2}$$

It is clear from eq.(1) that the value R is proportional to the average connection time. Therefore, the degradation of server performance can be evaluated with increase of R.

The benefits of employing R from eq.(2) are summarized as follows.

- The parameters that are used to compute R, are obtained through passive measurement of user traffic. This means that the metric R can be estimated without offering an additional load on the server or the network.
- The metric R can be easily computed and can be obtained without managing TCP connections. Thus, the amount of computational time and storage is much smaller for this metric than for the metrics used in previous works such as [3, 4, 13, 14].

3.2 Evaluation

To show the effectiveness of the proposed metric R, an experiment was executed. The experiment employed three Linux PCs (PC0, PC1, and PC2) connected by 1 Gb/s Ethernet, as shown in Fig. 1. The PCs used in the experiment are 3.06 GHz Celeron machines with Plamo Linux 4.1, Kernel 2.6.18.1. A WWW server (Apache) runs on PC0, while the client runs on PC2. The client program (`httperf` [15]) generates HTTP GET requests for 1×10^5 byte files on the WWW server. The interval between requests is randomly determined according to the exponential distribution. The connection time was measured by the client program, and the proposed metric R was measured on PC1. The obtained connection time and metric R were compared. The proposed metric R was measured using `tcpdump` [16]. That is, `tcpdump` was executed on PC1 during the measurement period to capture every packet whose SYN flag was set. By recording TCP SYN and SYN ACK packets in this manner, the numbers of TCP SYN and SYN ACK messages, N_S and N_A, were counted. Then, the metric R was computed by eq.(2).

The result is shown in Fig. 2. The x-axis is the traffic load represented by the TCP connection request rate, whereas the y-axis is the average connection time and the metric R. The figure shows that the characteristics of R almost exactly coincide with those of the connection time. Therefore, we can employ R rather than measuring the connection time, whose estimation is more difficult. It is also observed that performance degradation is identified correctly through the proposed metric R.

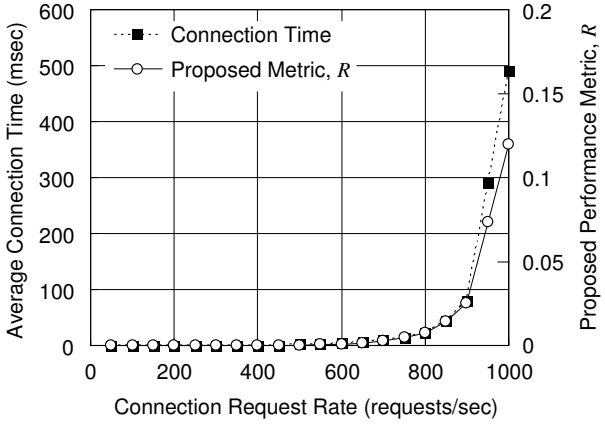

Fig.2: *Relationship between the proposed performance metric R and the average TCP connection time.*

Figure 2 shows that the average connection time is nearly 3000 times R. Thus, the above characteristics and eq.(2) imply that the timeout period of TCP SYN retransmission is about 3 seconds in the experimental environment. This result agrees completely with the initial timeout specified in Sect. 4.2.3 of Ref. [17]. This confirms that the metric R is determined by the mechanism described in Sect. 3.1 of the present paper.

In the Fig. 2, small differences between R and the connection time appear at certain points. These differences are caused by delay factors other than TCP SYN retransmission.

4. LOAD BALANCING TECHNIQUE

This section describes a load balancing technique in which the load balancer estimates the performance metric R for each server. If R is greater for one server than for others, the load balancer judges that machine to have a lower remaining capacity. Thus, the load balancer decreases the machine traffic. Details of the load control algorithm and some implementation issues are described below.

4.1 Algorithm

Suppose that n servers, $1, 2, \ldots, n$, offer the same services and share requests from the clients. Let r_i $(0 \leq r_i \leq 1)$ be the traffic ratio for server i $(1 \leq i \leq n)$, i.e., (traffic given to server i)/(total traffic). Moreover, assume the environment to be heterogeneous, and the capacities of some servers to be rela-

tively greater. Thus, it is necessary to feed different traffic loads to each server so as to maximize the total performance.

A greater value of the metric R means that the machine has less remaining capacity and that its performance is degraded. Thus, the average connection time will be improved by decreasing the load from such a machine. The actions of the algorithm are as follows:

1. Measure the numbers of TCP SYN and SYN ACK messages for each server periodically, and compute the metric R.
2. Select the server that has the largest value of R among servers $1, 2, \ldots, n$. Let j denote the selected server.
3. Decrease the traffic ratio r_j by δ $(0 < \delta < r_j)$, while increasing $r_i (i \neq j)$ by $\delta/(n-1)$.

4.2 Implementation

To implement the proposed algorithm, some issues must be addressed. The first is the manner in which the TCP SYN and SYN ACK messages are counted during an appropriate time period. If the numbers of TCP SYN and SYN ACK messages are estimated for a too long period of time, it becomes impossible to control traffic adequately against a rapid change of server performance. Thus, the counted number of these messages must be updated smoothly to reflect the changes in server performance against time. To achieve such an update, this study examines the following scheme.

Assume that the metric R is computed every T seconds from the numbers of TCP SYN and SYN ACK messages, arriving during the most recent mT seconds (m: integer). This means that R is computed from TCP SYN and SYN ACK messages that have arrived in the time window whose size is mT seconds. Let $s_0, s_1, \ldots, s_{m-1}$ be m counters to count TCP SYN messages, while $a_0, a_1, \ldots, a_{m-1}$ are m counters to count SYN ACK messages. In addition, let t denote the current time in seconds. Counter s_k (or a_k) is incremented by 1 for each TCP SYN (or SYN ACK) message arriving during the period,

$$(Nm + k)T \leq t < (Nm + k + 1)T, \tag{3}$$

where N is an integer. For example, if T is 10 seconds, s_0 is incremented for $0 \leq t < 10$, s_1 is incremented for $10 \leq t < 20$, and so on. Then, when the current time satisfies

$$t = (Nm + \hat{k})T, \tag{4}$$

for integer $\hat{k}$, the metric R for the most recent mT seconds is computed by

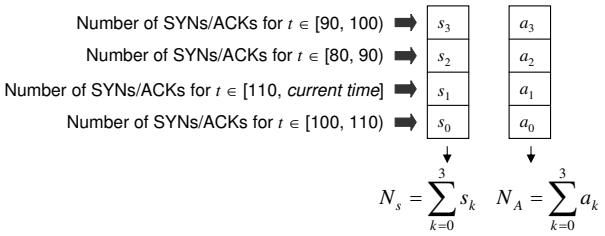

Fig. 3: *Counter configuration for calculating the metric R every 10 seconds from data obtained in a 40-second period.*

$$R = \frac{\sum_{k=0}^{m-1} s_k - \sum_{k=0}^{m-1} a_k}{\sum_{k=0}^{m-1} a_k}. \tag{5}$$

Simultaneously, $s_{\hat{k}}$ and $a_{\hat{k}}$, which hold the oldest data among the counters, are reset to zero. This procedure is illustrated by Fig. 3.

The size of the time window, mT, must be sufficiently large to obtain a reliable value of R. If the time-window size is too small, very few lost TCP SYN messages are counted during the time window. For such a case, it is difficult to obtain an adequate number of samples for reliable estimation of R.

The second issue is how the parameter δ should be chosen in the algorithm. Since the traffic ratio r_j cannot be less than 0, δ should not exceed r_j. Thus, this study examines the value

$$\delta = \Delta r_j \qquad (0 < \Delta < 1). \tag{6}$$

The value of Δ must be much smaller than 1. To understand this, let us suppose that the load is now shared among two server machines denoted by S_0 and S_1 and that the value of Δ is close to 1, for example, 0.99. Additionally, consider that the metric R for S_0 becomes larger than that for S_1. Then, the algorithm routes most of the traffic load to S_1. Since this degrades the performance of S_1, R will become larger for S_1 than for S_0. Thus, at the next control event, most of traffic load will be routed to S_0. This means that the loads on S_0 and S_1 oscillate with the period of the control event interval. Needless to say, such oscillation is undesirable and thus must be avoided by setting Δ to a sufficiently small value.

Meanwhile, a small Δ will slow down the response of the control against performance changes because the load will change very slightly at each control event. Although further study is needed to find an optimal value of the parameter Δ, the experimental result suggests that $\Delta = 0.1$ is a good setting.

It is easy to distribute requests to server i according to r_i, by employing the `iptables` command of the

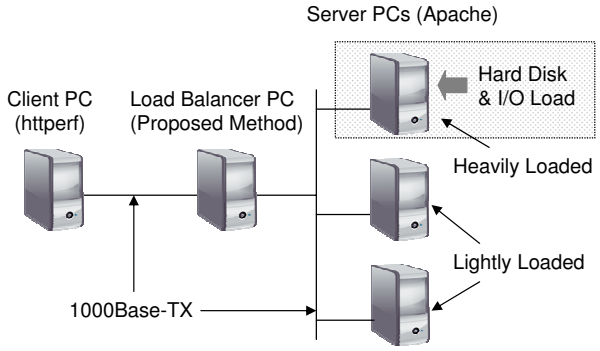

Fig.4: *Network configuration evaluating the proposed load balancing method.*

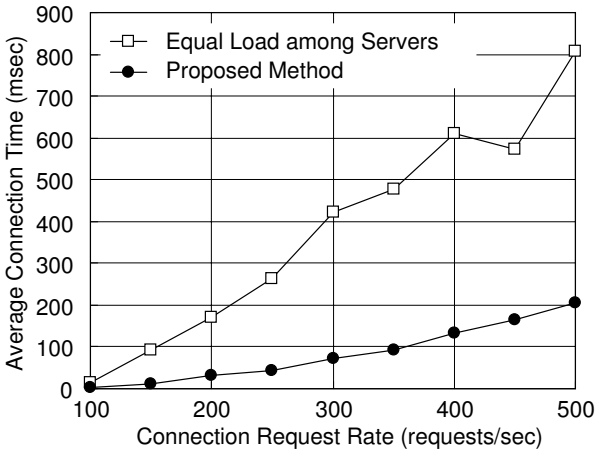

Fig.5: *The average connection time versus the connection request rate for the proposed method and the equal load case.*

Linux OS. The `iptables` command supports Destination Network Address Translation (DNAT) suited to load balancing [18]. Moreover, the load for each server can be controlled using the "statistic" extension and its "probability" option [19]. Thus, the load is exactly divided among each server by calculating the probability given to the `iptables` rule from r_i.

4.3 Evaluation

The effectiveness of the presented load balancing technique was evaluated through an experiment performed on the network as shown in Fig. 4. The network is configured with three server PCs, a load balancer PC, and a client PC. These PCs are connected by two local networks (1 Gb/s Ethernet). The Apache WWW server was running on the server PCs, while `httperf` was executed on the client PC to generate HTTP GET requests and measure the performance. The proposed technique was executed on the load balancer PC for distributing the requests to the server PCs. The technique was implemented as a C language program that utilizes the `pcap` library [16] to capture TCP SYN and SYN ACK packets from user traffic. When the program receives a TCP SYN (or SYN ACK) packet, it updates the counter s_k (or a_k) as described in Sect. 4.2. Meanwhile, it computes the metric R every T seconds and modifies the traffic ratios $r_1, r_2, \ldots, r_n$ by executing the `iptables` command.

The experiment was performed for a situation in which server performances differ greatly. This situation is generated by running a `stress` tool [20] on one server PC and generating excessive hard disk and I/O load. By executing `httperf` on the client PC, HTTP GET requests are issued for a 1×10^5 byte HTML file on the servers in a random interval determined according to the exponential distribution. The average connection time for this setting was measured by `httperf`, for the case applying the proposed method as well as for the case in which the traffic was distributed to servers with equal probability. The connection time was measured for 10^5 TCP connections generated by `httperf`. This measurement was repeated 15 times for each connection request rate, and then the average connection time was obtained. The initial value of r_i was 1/3 for every server PC. The parameters Δ, m, and T, used in the proposed method (see Sect. 4.2), were set at 0.1, 10, and 20, respectively. This means that the traffic ratio for each server is updated every 10 seconds on the basis of TCP SYN and SYN ACK messages counted for the most recent 200 seconds. In addition, the traffic for the most deteriorated server is decreased by 10 % when the traffic ratio is updated. The value of Δ was set sufficiently small to avoid load oscillation as described in Sect. 4.2. The time-window size, 200 seconds, was chosen to obtain an adequate number of lost TCP SYN messages for reliable estimation.

In the experiment, the PCs for the client, the load balancer and the heavily loaded server are 3.06 GHz Celeron machines. Meanwhile, the PCs for the lightly loaded servers are 1.60 GHz Celeron machines. Although the hardware specification of the heavily loaded server is superior to that of the lightly loaded server, its performance is worse because of the artificial load offered by `stress`. The OS employed in the PCs is Plamo Linux 4.1, Kernel 2.6.18.1.

Figure 5 shows the average connection time against the connection request rate. Evidently, the connection time is shorter for the proposed method than for the equal-load case. This result is obtained because the proposed method effectively reduces the traffic for the server degraded by a heavy load.

Figure 6 depicts the changes in traffic ratio r_i assigned to the server PCs against time. The figure shows that the load on the degraded PC effectively decreases with time. This means that most requests are sent to the server PCs with a light load, and thus do not suffer from serious connection delay. In addition, the performance of the degraded server is also

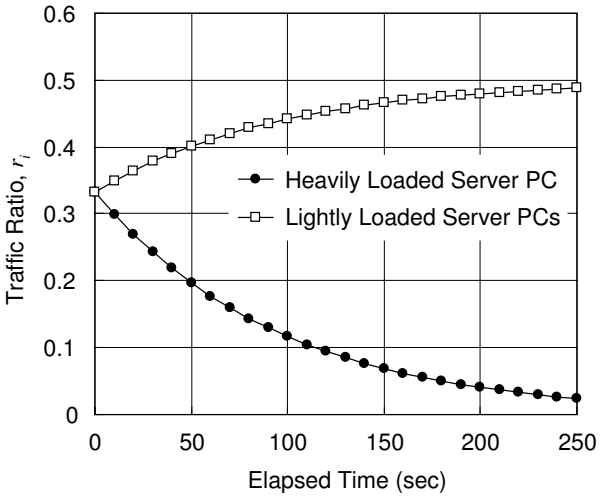

Fig.6: *Traffic ratio change against time.*

improved because of the decreased load. As a result, the average connection time is improved, as shown in Fig. 5.

The above result confirms that the proposed method successfully improves performance by controlling the traffic load on servers when the performance of a few servers displays relative degradation.

5. CONCLUSION

This paper presented a load balancing technique for WWW servers conducting a passive measurement of server performance. Initially, the paper described a server performance metric, estimated by passive measurement. The experimental result confirmed that the proposed metric correctly indicates the connection time increase brought about by server degradation. Further, a load balancing technique utilizing the metric was proposed. The experimental result confirmed that the presented load balancing technique effectively improves the performance.

References

[1] H. Bryhni, E. Klovning, and Ø. Kure, "A comparison of load balancing techniques for scalable Web servers," IEEE Network, 14, 4, pp.58-64, July/Aug. 2000.

[2] T. Bourke, Server load balancing, O'Reilly, Aug. 2001.

[3] H. Miura and M. Yamamoto, "Content routing with network support using passive measurement in content distribution networks," IEICE Trans. on Communs., E86-B, pp.1805–1811, June 2003.

[4] U. Lee, J.-s. Park, M.Y. Sanadidi and Mario Gerla, "Flow based dynamic load balancing for passive network monitoring," in proc. Communications and Computer Networks 2005, pp.357-362, Marina del Rey, CA, Oct. 2005.

[5] The Linux Virtual Server Project, http://www.linuxvirtualserver.org/.

[6] W. Zhang, "Linux virtual servers for scalable network services," in proc. Ottawa Linux Symposium 2000, Ottawa, July 2000.

[7] G. Goldszmidt and G. Hunt, "NetDispatcher: a TCP connection router," IBM Research Report, RC 20853, May 1997.

[8] J. Kerr, "Using dynamic feedback to optimize load balancing decisions," in proc. Australian Linux Conference 2003, Perth, Australia, Jan. 2003.

[9] C.-h. Tsai, K.G. Shin, J. Reumann and S. Singhal, "Online web cluster capacity estimation and its application to energy conservation," in proc. IM2005, Application Sessions, Session 5, Nice, France, May 2005.

[10] C.-H. Tsai, K.G. Shin, J. Reumann and S. Shinghal, "Online web cluster capacity estimation and its application to energy conservation," IEEE Trans. Parallel and Distributed Systems, 18, 7, pp.932–945, July 2007.

[11] P. Pradhan, R. Tewari, S. Sahu, A. Chandra and P. Shenoy, "An observation-based approach towards self-managing web servers," in proc. IWQoS 2002, pp.13-20, Miami Beach, May 2002.

[12] T. Nakasima and T. Sueyoshi, "Performance estimation of TCP under SYN flood attacks," in proc. CISIS'07, pp.92-99, Vienna, Austria, Apr. 2007.

[13] Y.-d. Lin, S.-c. Tsao and U.-p. Leong, "On-the-fly TCP path selection algorithm in access link load balancing," Computer Communications, 30, 2, pp.351–357, Jan. 2007.

[14] C.E. Chow and C. Prakash, "Enhance features and performance of a Linux-based content switch," in proc. IASTED Conf. on Applied Informatics, Innsbruck, Austria, Feb. 2003.

[15] D. Mosberger and T. Jin, "httperf – a tool for measuring web server performance," ACM SIGMETRICS Performance Evaluation Review, 26, 3, pp.31-37, Dec. 1998.

[16] TCPDUMP/LIBPCAP public repository, http://www.tcpdump.org/.

[17] R. Braden, "Requirements for Internet hosts – Communication layers," IETF RFC 1122, Oct. 1989.

[18] R. Russel, "Linux 2.4 NAT howto," http://security.maruhn.com/nat-howto/.

[19] Linux iptables libxt_statistics Manual.

[20] Stress project page, http://weather.ou.edu/ apw/projects/stress.

Satoru Ohta received the B.E., M.E., and Dr. Eng. degrees from the Tokyo Institute of Technology, Tokyo, Japan, in 1981, 1983, and 1996, respectively. In 1983, he joined NTT, where he worked on research and development of cross-connect systems, broadband ISDN, network management, and telecommunication network planning. Since 2006, he has been a professor with the Department of Information Systems Engineering, Faculty of Engineering, Toyama Prefectural University, Imizu, Japan. His curret research interests are performance management of information networks and network protocols. He is a member of the IEEE and IEICE. He received the Excellent Paper Award in 1991 from IEICE.

Ryuichi Andou recieved the B.E. degree from Toyama Prefectural University, Imizu, Japan, in 2008. He is now a master course student in the Department of Information Systems Engineering, Faculty of Engineering, Toyama Prefectural University. His research interests are network monitoring and network control.