Request Distribution for Fairness with a New Load-update Mechanism in Web Server Cluster

MinHwan Ok[1], Myong-soon Park[2]

[1]Korea Railroad Research Institute
Uiwang, Gyeonggido, 437-050, Korea
panflute@korea.ac.kr
[2]Dept. of Computer Science and Engineering, Korea University
Seoul, 136-701, Korea
myongsp@ilab.korea.ac.kr

Abstract. The complexity of services and applications provided by Web sites is ever increasing as integration of traditional Web publishing sites with new paradigms, i.e., e-commerce. Each dynamic Web page is difficult to estimate its execution load even with the information from the application layer. In this paper the execution latency at Web server is exploited in order to balance loads from dynamic Web pages. With only the information such as IP address and port number for Layer-4 Web switch the proposed algorithm balances the loads with a new load-update mechanism. The mechanism uses the report packets more efficiently with the same communication cost. Moreover the proposed algorithm considers the fairness for Web clients hence the Web clients would experience higher quality of service.

1. Introduction

Web service is the most prevalent Internet service and its importance and usage gets higher as years go. For large numbers of Web client requests are headed to a popular Web site in peak times, most of the sites form multiple nodes into one Web server cluster. In these systems, any client Web request to the system is presented to a front-end server that acts as a representative for the system. This is called Web switch retains transparency of the parallel architecture for the user, guarantees backward compatibility with Internet protocols and standards, and distributes all Web client requests to the back-end Web servers. Web server cluster in this paper collectively indicates this formation of a Web switch and Web servers, as illustrated in Fig. 1. The Web switch should distribute incoming requests to Web servers in load-balanced fashion. With only the information such as IP address and port number it seems some limit exits to develop request distribution algorithm load-balancing. Moreover most of Web pages are incorporated with executing scripts such as Java, PHP and so on, load-balancing becomes much difficult with requests for those dynamic Web pages. Due to variant execution latencies of the executing scripts, load-balancing in distributing requests for dynamic Web pages need consider fairness among Web clients. In this paper we propose a request distribution algorithm with a new load-update mechanism.

Although so many algorithms have been proposed, in our best knowledge, no previous work has suggested the load-update mechanism. In the next section the kinds of information for load-balancing is introduced and the fairness among the Web clients is described. The load-update mechanism is suggested in developing a request distribution algorithm in Section 3. The proposed algorithm is compared with the related previous works in a sense of load-update mechanism in Section 4 and the simulation results are presented in Section 5. The last section concludes with the effect of the new load-update mechanism.

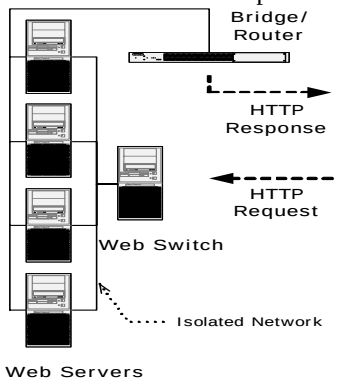

Fig. 1. Web Server Cluster with Isolated System Network

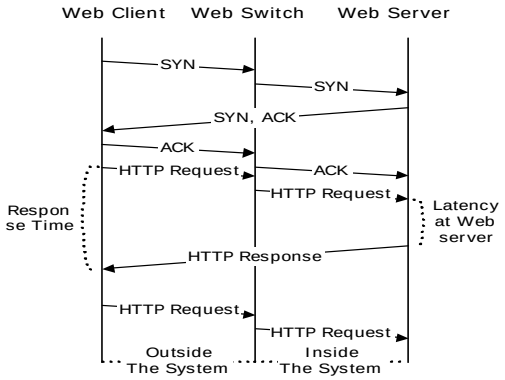

Fig. 2. Web Service Protocol of the Cluster

2. Web Switch and Request Distribution for Fairness

For the performance feature of the Web server cluster the Web switch should distribute the requests to the Web servers so the loads of Web servers are balanced in the cluster. According to the OSI protocol stack layer at which the Web switch operates, Web switches are broadly classified into Layer-4 and Layer-7 Web switches.[1] The Layer-4 Web switch has the only information related to TCP/IP layers, thus the information such as IP address and port number. The Layer-7 Web switch parses the request then gets information of up to the application layer, thus the information such as URL contents, SSL identifiers and cookies. The Layer-4 Web switch is not aware of content information whereas the Layer-7 Web switch is. As much information is supplied, the Layer-7 switch is capable for more accurate decision in distributing the requests. However the Layer-7 Web switch introduces severe processing overhead to the extent that may cause the Web switch to severely limit scalability of the Web server cluster. In [5], the peak throughput achieved by a Layer-7 Web switch is limited to 3,500 connections per second, while software based Layer-4 Web switch implemented on the same hardware is able to sustain a throughput up to 20,000 connections per second. For this reason we propose a distributing algorithm and the system organization found on the Layer-4 Web switch.

Many works on load balancing in distributing requests are conducted for the Web server cluster.[2, 3, 4] The works considers mainly the static Web pages rather than

the dynamic Web pages with executing scripts. Nowadays there are much of dynamic Web pages of Java, PHP, ASP and so on. Those Web pages are difficult to estimate the load even with the information from the application layer. Although there might exist such an estimation algorithm, the Web switch cannot use highly sophisticated algorithms in distributing requests since it has to take immediate decision for hundreds or thousand of requests per second. We need a simple algorithm for this reason.

Typical service protocol of the Web server cluster is illustrated in Fig. 2. When the Web switch received the Web client request, it determines whether the request is from current connected Web client or for new connection to a Web server. In the case the request is of current connection by a hash function at Web switch, the request is relayed to the connected Web server. Otherwise, the distribution algorithm selects a Web server for new connection. After processed at the Web server the response of the request is routed to the Web client. As depicted in Fig. 2, the response time is composed of four parts;

$$t_{Response} = t_{Outside} + t_{Distribution} + t_{Inside} + t_{Processing} \quad (1)$$

$t_{Response}$ is the time elapsed after it send its request till the Web client starts receiving the Web server's response. The client request takes a half of $t_{Outside}$ to reach the Web switch, and the server response takes the other half of $t_{Outside}$ to reach the client. This is the time elapsed outside the system only. The Web switch received the request decides the Web server by the distribution algorithm or a hash function for $t_{Distribution}$. The request is then relayed to the Web server in a half of t_{Inside} and processed at the Web server for $t_{Processing}$. After processed the response takes the other half of the t_{Inside} to leave the system, in other words, to reach the bridge/router in fig.2. Among those parts, $t_{Outside}$ is dependent on the location of the Web client. Thus it is variant and unable to reduce at the system. Reducing t_{Inside} is not relevant to the distribution algorithm thus out of the scope of this paper.

To reduce $t_{Distribution}$, the latency at Web switch it is necessary a fast and simple distribution algorithm. Balancing the loads of each Web server would reduce $t_{Processing}$, the latency at Web server, although it is directly dependent on the Web server's capacity. The latency at the Web switch is common for every Web client. Thus equalizing the average of latencies at the Web servers should be fair for the Web clients with respect to the response time. For $t_{Outside}$ is variant by each Web client's distance from the system, the response times are not equal in the reality, however we appreciate it is also fair for all the clients. Therefore we use the execution latencies for comparison of each server's load to evaluate the load-balancing. Exploiting the response time is nothing new for load balancing[6], however the latency at Web server is not exactly the same with the response time. Moreover distribution algorithm we propose has a particular load-update mechanism. The new algorithm works far differently from other algorithms proposed so far.

3. Load-Balancing with a New Load-update Mechanism

Conventionally the Web switch of the cluster gathers information of each server's load periodically. All the servers in the cluster reports their load information to the Web switch at any given rate, and the load-balancing algorithms used this *periodic load-update* mechanism. This mechanism is good for updating simultaneously actual loads of all the servers since the reporting is synchronized by all the servers. However this mechanism is not good for system scalability since the number of report packets concentrated to the Web server grows as the number of the servers increases. In this section we introduce a new load-update mechanism that the reporting is not synchronized among the servers.

The objective of load-balancing algorithm is to keep even loads among the servers. Previous studies have suggested that the run-queue length best describes a server's load, and many load-balancing algorithms have adopted this metric[9]. We focus on execution latencies thus we adopt this metric. Our basic idea is once sending equal numbers of requests to each server and then let the server having less requests report its 'lessness'.

Load-balancing Algorithm 1. *High-Communication-Cost Model*

```
For every request packet arriving at the Web switch;
1. The Web switch merely distributes the income
requests to servers in traditional 'Round-Robin'. Equal
numbers of requests are executing in the servers.
2. When a request finishes its execution the server
that processes the request immediately reports that one
request has finished.
3. The Web switch subtracts one from the load value of
the reported server in the Load Table.
4. IF the load values are not all equal the Web switch
finds the lowest value and sends one request to the
server,
  ELSE the Web switch sends the request in 'Round-Robin'
order.
5. Whenever the Web switch sends one request, it adds
one to the load value of the target server in the Load
Table. Continue at Line 2.
```

Line 4 of the algorithm guarantees the Web switch keep the numbers of executing requests equal among servers. This algorithm is quite simple and works nicely. There are two conditions of early finish; the execution length of the request was shorter in itself, or the request shared resources with fewer other requests, i.e. CPU. While most of other requests are in IO-phase the requests in CPU-phase gets more CPU times. Each server is processing equal number of requests at any instant, however the throughput of each server is different. Since the Web switch does not have enough

time to reschedule income requests considering efficient overlap of one request's CPU-phase and other request's IO-phase, this algorithm should show ideal load-balancing. The algorithm is compared with previous works in Section 4.

For the algorithm to work each request's finish must be immediately reported to the Web switch. We named this load-information update mechanism *Update-on-Finish*. This update is neither periodic nor synchronized among servers. If the requests received from the Web switch are n requests, the reports should be sent exactly n times. Although the 'isolated network' of the Fig. 1 could accommodate the communication for reporting, communication cost to the Web switch in this number of updates may be high. We extend the algorithm for less costly communication.

Algorithm 1 ensures all the servers keep the numbers of executing requests equal among servers. At each server, while some of requests finish the execution, new requests are arriving by Round-Robin of Algorithm 1. Thus the reporting is only necessary when the number of executing requests decreases. The Web switch distributes requests in Round-Robin or sends more requests to the server when the report comes.

Load-balancing Algorithm 2. *Lower-Communication-Cost Model*

For every request packet arriving at the Web switch;

1. The Web switch merely distributes the income requests to servers in traditional 'Round-Robin'. Equal numbers of requests are executing in the servers.

2. When the number of executing requests decreases the server reports that n requests are more needed.

3. The Web switch subtracts n from the load value of the reported server in the Load Table.

4. IF the load values are not all equal the Web switch finds the lowest value and sends one request to the server,

   ELSE the Web switch sends the request in 'Round-Robin' order.

5. Whenever the Web switch sends one request, it adds one to the load value of the target server in the Load Table. Continue at Line 2.

In Algorithm 1, each server receives one more request instantly after the server reported that it has requests one less than other servers. However the Web switch is distributing requests in Round-Robin otherwise, the number of executing requests soon recovers after one request has finished. Recall that the execution lengths of requests are not same each other. Receiving the equal number of requests does not result in the equal number of executing requests.

Let Φ be the period of Round-Robin. Assume one request has finished execution at a server and the server should receive one request within $\Phi/2$ by Round-Robin. Let the server do not report, if no more requests finish within $\Phi/2$. The server

counts the number of executing requests at every $\Phi/2$. Thus the server reports after one or more reports finished within the first half of Φ, and two or more requests finished within the second half of Φ. If the server counts the number of executing requests at every Φ, the line 2 of Algorithm 2 reduces the communication cost to $1/m$ when mean m execution finishes are reported in each packet.

4. Related Works and Comparison

Many experiments and simulation results have demonstrated that the Weighted Round-Robin (WRR) comprises simplicity with efficacy at best[2]. Most recent work exploited load-update mechanism is Dahlin's algorithm[7]. WRR uses periodic load-update. Once Web switch realized each server's load, it sends requests to a less loaded server with higher rate and sends requests to a more loaded server with lower rate until they reach equal loads before next load-update. Dahlin's algorithm also uses periodic load-update. The web switch realizes the differences in loads between servers by load-update. It sends requests to servers with least loads. After all other servers' loads are equalized to the most loaded server, the Web switch distributes requests in Round-Robin manner before next load-update. Now we compare the proposed algorithms to these two algorithms with respect to load-update mechanisms.

For any algorithm, higher rate of load-update achieves more balanced load distribution between servers. We define reporting cost, R, as follows;

R : the number of packets received by the Web switch for a given period

Thus reporting cost of periodic update, $R_P = n \cdot p$, where n is the number of servers and reporting are p times in the period. While exactly n report packets should be used at every reporting time in periodic update mechanism, with the same reporting cost of Update-on-Finish (in Algorithm 2), $R_U = R_P$, the server use the report packet only when the number of executing requests decreases. Whereas each server uses exactly p packets during a given period in periodic update mechanism, the Update-on-Finish mechanism (in Algorithm 2) allow more reporting for the servers that finish requests more frequently, and less reporting for the servers that finish requests less frequently with $n \cdot p$ packets. Therefore Update-on-Finish mechanism uses the report packets more efficiently with the same communication cost. In the next section we compare the proposed algorithm with the two algorithms.

5. Performance Evaluation

A simulator of Web server cluster is implemented. A Web switch and 5 Web servers constitutes the cluster. 4000 requests are processed for simulation of 10 seconds. The execution lengths of the requests range from 5 to 50 and each server processes 0.2 (in length) of a request per millisecond. The generation of execution lengths follows Pareto distribution. Pareto distribution have been found to correspond to some real

world workloads such as a Web request's execution length[2]. We model the execution lengths as being generated independently and identically distributed from a distribution that follows a power law, but has an upper bound. It is characterized by three parameters: α, the exponent of the power law; k, the smallest possible observation, and p, the largest possible observation. The probability mass function of this Pareto distribution is defined as:

$$f(x) = \frac{\alpha k^{\alpha}}{1-(k/p)^{\alpha}} x^{-\alpha-1}, \quad k \le x \le p. \tag{2}$$

In most cases where estimates of α were made, α tends to be close to 1, which represents very high variability in service requirements. It is known that Poisson distribution is far from realistic for request arrival through Internet. Request arrival process follows uniform distribution.

Weighted Round-Robin, Dahlin's, and Load-Balancing Algorithm 2 were simulated with the same reporting costs. The reporting costs are averaged from one hundred generations of 4000 request packets. Algorithm 2 was simulated first with one hundred generations to reckon up the number of report packets sent from the servers. Then we found equivalent update periods to the average numbers of report packets as corresponded in Table 1.

Table 1. Corresponding update periods with equal reporting costs

Number of report packets	1199	1540	2086
Update periods	42 *msec*	32.5 *msec*	24 *msec*

WRR shows 3.82 as mean execution latency of 5 servers with 1199 report packets in Fig. 3. Dahlin's shows 2.91 as mean execution latency of 5 servers with 1199 report packets in Fig. 4. Algorithm 2, the proposed load-balancing, shows 2.86 as mean execution latency of 5 servers with 1199 report packets in Fig. 5. Algorithm 2 has the lowest standard deviations(Sum of standard deviations - Dahlin's: 6.30893; Algorithm 2: 5.69997) among the three algorithms. WRR and Dahlin's performed using periodic update with the period of 42 milliseconds(equivalent to 1199 report packets). With the equal reporting cost, Algoritm 2 outperformed and the gap is ever increasing as more report packets are used. Figure 6 illustrates the effect of the update period.

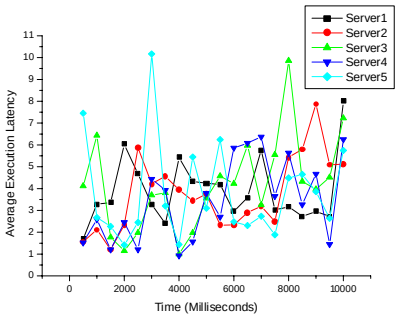

Fig. 3. Execution Latencies of Requests in WRR

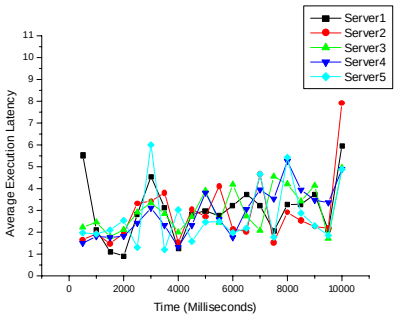

Fig. 4. Execution Latencies of Requests in Dahlin's

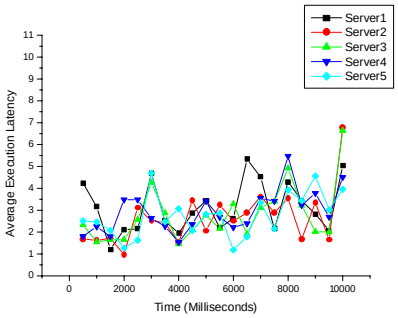

Fig. 5. Execution Latencies of Requests in Algorithm 2

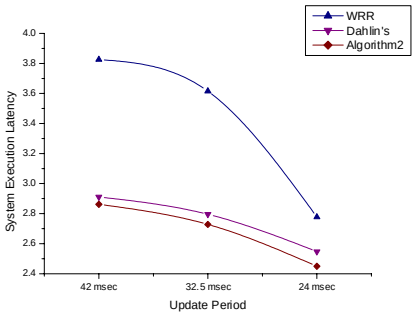

Fig. 6. Mean Execution Latency of 5 Servers, which the Web Clients Experience

The values of the figure are averaged from one hundred simulations. The mean execution latency of 5 servers, system execution latency in the figure, reduces slightly more than Dahlin's algorithm as the update period decreases. The Web switch sends reciprocally proportional numbers of packets according to each server's load for a period to balance loads of the servers until next load-update in WRR. Dahlin's balances the server loads as soon as possible with current load information, and then the next packets distributed by the Web switch are sent in Round Robin until next load-update. Since Dahlin's acquires load-balancing much earlier than WRR, the gap between the two algorithms is large. Algorithm 2 balances the loads whenever a report packet arrives, keeping request packets sent for load-balancing in the Load Table. Each server sends the request packet when it needs more request packets for a period whereas Dahlin's use report packets at mandatory update time. Thus actions for load-balancing happens more times than Dahlin's and this difference results in the gap between Dahlin's and Algorithm 2.

6. Conclusion

The requests for dynamic Web pages are difficult to estimate the loads as executing scripts have variant execution lengths, and it becomes much variant if the Web page gets input parameters for the executing scripts. With only the information such as IP address and port number for Layer-4 Web switch the proposed algorithm balances the loads by sending packets as needed. Report packets contains the number of packets finished their executions. Thus the load-update is naturally non-periodic. The proposed algorithm showed 98.32 percent, 97.57 percent, and 96.15 percent of Dahlin's algorithm in system execution latency with reporting costs equivalent to the update periods of 42, 32.5 and 24 milliseconds, respectively. Moreover the proposed algorithm considers the fairness for Web clients hence the Web clients would experience higher quality of service. Another advantage of the algorithm is its simplicity, since simpler distribution algorithm leads to higher throughput of the Web switch.

Although not resented in this paper the non-periodic load-update occurs asynchronously among servers. This reduces the communication workload for the Web switch than that of periodic update since all the report packets are not concentrated at any instant. Thus the new load-update mechanism would support higher scalability.

References

1. T. Schroeder, S. Goddard, B.Ramamurthy. "Scalable Web server clustering technologies", IEEE Network, May-June 2000, pp. 38-45
2. V. Cardellini, E. Casalicchio, M. Colajanni, P. Yu. "The state of the art in locally distributed Web-server systems", ACM Computing Surveys, 34(2), June 2002, pp. 263-311
3. M. Andreolini, M. Colajanni, R. Morselli. "Performance study of dispatching algorithms in multi-tier Web architectures", ACM SIGMETRICS Performance Evaluation Review, 30(2), September 2002, pp. 10-20
4. E. Casalicchhio, M. Colojanni. "A client-aware dispatching algorithm for Web clusters providing multiple services", 10th International World Wide Web Conference, May 2001, pp. 535-544
5. M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel. "Scalable content-aware request distribution in cluster-based network servers", The 2000 USENIX Annual Technical Conference, June 2000
6. Cisco Systems Local Director. http://www.cisco.com, 2002
7. M. Dahlin. "Interpreting stale load information", IEEE Trans. Parallel and Distributed Systems, 11(10), October 2000, pp. 1033-1047
8. http://support.zeus.com/doc/tech/linux_http_benchmarking.pdf
9. T. Kunz, "The Influence of Different Work-load Descriptions on a Heuristic Load Balancing Scheme:, IEEE Trans. Software Eng., Vol. 17, No. 7, July 1991, pp. 725-230