System Support for Workload-aware Content-based Request Distribution in Web Clusters

Yu-Chen Lin Mei-Ling Chiang Lian-Feng Guo

Department of Information Management
National Chi-Nan University, Puli, Taiwan, R.O.C.
Email: {s2213507, joanna, s3213534}@ncnu.edu.tw

Abstract

We have designed and implemented a high-performance and scalable content-based web cluster named LVS-CWARD. The LVS-CWARD cluster uses a small amount of memory dedicated to cache a small set of most frequently accessed files and uses a kernel-level content-based web switch to distribute the HTTP requests from clients among the web servers. The proposed CWARD policy which takes into account the content of requests and workload characterization in request dispatching, with the aim to increase cluster performance by increasing the cache hit rates in web servers. Besides, to efficiently support for persistent connection, the fast Multiple TCP Rebuilding is proposed. Moreover, we propose two request scheduling algorithms to achieve more performance gain by more fine-grained loading measurement for more effective load sharing. The experimental results of practical implementation on Linux show that our proposed kernel-level web switch which takes into account the content in requests and workload information in dispatching requests is efficient and scales well without being the system bottleneck of the whole cluster. Moreover, the trace-driven benchmarking with the real-world working sets demonstrates that our proposed system can achieve up to 107% and 164% better performance respectively than the Linux Virtual Server with a content-blind web switch in distributing requests.

Keywords: Content-based Web Cluster, Web Cluster, Content-aware Web Switch, Content-blind Web Switch, Persistent Connection

1. Introduction

To deal with the explosive growth of the World Wide Web, locally distributed web-server systems are the most popular configuration rather than the single scale-up web-server systems. In the distributed web-server systems, the cluster-based web systems have been widely adopted because they are transparent and well balanced. A state-of-the-art cluster-based web system (briefly, web cluster) employs a web-switch called front-end which distributes requests from clients among the request handling servers called back-ends to achieve load sharing and scalability.

The front-end of a web cluster could be classified as layer-4 or layer-7 web switch according to the operating layer in the OSI protocol stack. Basically, the layer-4 web switch dispatches the requests based on the IP address and TCP port, whereas, the layer-7 web switch could perform the content-aware dispatching which dispatches the requests in accordance with the content (i.e. URI) examined from the requested packets.

Recently, more and more studies [1,2,4,6,7,8] focus on the content-aware dispatching and their results show that using the content of requests and loading information, a web cluster will be more efficient in handling all types of requests. For example, Aron et al. proposed a novel content-aware dispatching policy named LARD [7] which aims at optimizing the usage of the overall cluster RAM, thus achieving the better cache hit ratio in back-ends. Subsequently, Cherkasova et al. proposed the WARD strategy [2] which decreases the overhead incurred from TCP handoff [7] using LARD policy while still optimizing the overall cluster RAM. Besides, using content-aware dispatching, partitioning web contents, building specialized web services among web servers, or maintaining session integrity can be achieved.

In our previous study, Liu and Chiang proposed the TCP-Rebuilding technique [6], a light-weight TCP connection transfer technique that enables a web cluster to be content-aware. TCP Rebuilding could rebuild the TCP connection at one request-handling server using only the HTTP request packet and no extra packets for connection transfer are required. Therefore, we adopt the TCP Rebuilding to construct the content-aware platform in this research.

In this paper, we have designed and implemented a high-performance and scalable content-based web cluster named LVS-CWARD. The LVS-CWARD cluster uses a kernel-level layer-7 web switch and uses a small amount of memory dedicated to cache a small set of most frequently accessed files, with the aim to increase cluster performance by increasing the cache hit ratios in servers. We also propose a content-based workload-aware request distribution policy called CWARD, which takes content in requests and workload into account in distributing requests. Besides, to efficiently support

for persistent connection, the fast Multiple TCP Rebuilding is proposed and presented. Moreover, we propose two request scheduling algorithms to gain better performance by achieving more fine-grained load sharing.

The experimental results of practical implementation on Linux show that our proposed kernel-level web switch which takes into account the content in requests and workload information in dispatching requests is efficient and scales well without being the system bottleneck of the whole cluster. Moreover, the trace-driven benchmarking of real-world working sets [9] demonstrates that our proposed system can achieve up to 107% and 164% better performance than the layer-4 LVS web cluster [5] respectively.

2. Background

2.1 Linux Virtual Server

The Linux Virtual Server (LVS) [5] is a highly scalable and available server built on a cluster of real servers, with a load balancer running the Linux operating system. The architecture of the server cluster is fully transparent to end users, and the users interact as if it was a single high-performance virtual server.

The architecture of LVS comprises a front-end server (FE) and several back-end real servers (BEs). The front-end server is a load balancer responsible for dispatching and routing requests from clients to the real servers. The real servers handle requests and respond to clients. The cluster system is transparent to clients as a virtual service using a single IP address.
LVS has three routing mechanisms, i.e. NAT (Network Address Translation), IP tunneling, and direct routing [5]. The most efficient one is the direct routing mechanism, in which the FE forwards the packets from clients to chosen BEs by changing the MAC addresses of packets.

The front-end server in LVS is a content-blind load distributor that could not perform the content-aware distribution. However, because content-aware distribution can achieve higher cache hit rates in back-end servers and can apply sophisticated load-balancing strategy which could gain higher performance, thus, content-aware distribution is getting more and more popular in recent years.

2.2 TCP Rebuilding

TCP Rebuilding [6] is an efficient technique proposed in our previous study; it does not need the overheads of spoofing packets and processing of packet filter for TCP connection state transfer. As shown in Figure 1 [6], the packet forwarding steps are described as below:

1~4. The front-end performs the TCP three-way handshake with the client and then forwards the HTTP request from the client to the chosen back-end.

5. When the back-end receives the HTTP request, it rebuilds the TCP connection with the client using TCP Rebuilding technique [4].

6~7. After the connection has been rebuilt, the back-end could respond the requested data to the client directly.

8~9. When the front-end receives the ACK packet sent from the same client, it forwards it to the same back-end.

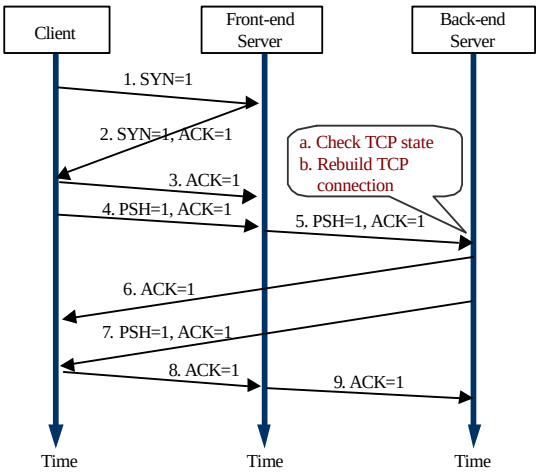


Figure 1: Packet Forwarding Flow of TCP Rebuilding

TCP Rebuilding belongs to one-way technique in which response packets from back-end can be sent directly to clients. Besides, it rebuilds the connection with client without the need of extra packet [7] or packet filter [4] for TCP connection state transfer. Therefore, TCP Rebuilding technique is applied in this research.

3. System Design and Implementation

3.1 Content-based Workload-aware Request Distribution – CWARD

Several researches [1,2,6,7,8] show that content-aware request distribution that takes content in requests into account in dispatching requests can make the resource utilization of web cluster more effective. In fact, being aware of workload information also helps in request dispatching [2].

In Arlitt and Williams's study of web server workload, they identified ten common characteristics in their collected data sets. One important characteristic is the high concentration of references in the web server. In their study, when caching 10% of the most frequently accessed files with the cost of

only 6-45 MB main memory size, it can achieve 80-96% cache hit ratio. Moreover, he noticed another interesting trace characteristic: small documents tend to be accessed much more frequently than larger documents. The above two studies conclude that caching a small set of most frequently accessed files may substantially increase the server cache hit ratio with the cost of only a small amount of main memory size.

WARD strategy [2] also takes advantage of workload characterization that assigns the most frequent files to be served by each back-end node locally to minimize the forwarding overhead incurred from TCP handoff for those most frequent files. Since the workload characterization of web traffic highly influences the performance of web service, thus, workload characterization should be taken into consideration when we design a web cluster.

In this paper, we have designed and implemented a web cluster with content-based and workload-aware request distribution, called LVS-CWARD, which adopts WARD strategy and uses a kernel-level layer-7 web switch (front-end) to dispatch requests. Besides, a small set of most frequently accessed files is prefetched into server's RAM to increase the performance of the whole web cluster. With a kernel-level layer-7 front-end using effective content-based request dispatching policies, the web cluster could achieve more load-sharing than a layer-4 front-end or RR-DNS could. The web contents are prefetched into server's RAM because web requests tend to request a whole file, whereas, the buffer cache of traditional file system caches the individual blocks of a file but not the whole file in the RAM. With the file prefetching method, we could make sure the whole file would be cached in the RAM. Prefetching web contents could avoid data transferring between RAM and disk for those prefetched files, which can decrease the disk access overhead.

The proposed Content-based Workload-aware Request Distribution (CWARD) policy in our system is shown in Figure 2. Similar to the WARD strategy, we identify a small set of most frequently accessed files, named core, and identify a set of less frequently accessed files to be partitioned among back-end nodes, called part. Each back-end prefetches identical core files and the exclusive part files in the RAM. The content-aware dispatcher examines the content information (i.e. URI) in each request and then forwards the requests to the corresponding back-end node.

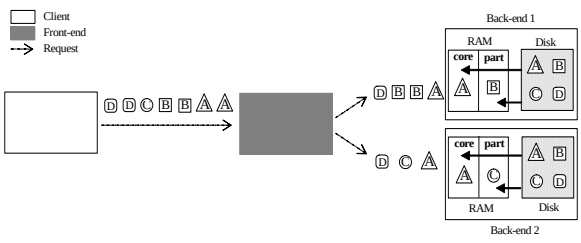


Figure 2: Dispatching Policy of CWARD

The CWARD policy shown in Figure 2 works as follows. First of all, the core files and part files are prefetched into server's RAM of back-end nodes and then the URL table in the front-end is updated with the corresponding information. When receiving a sequence of requests, the front-end will examine the content of each request. If the requested web file belongs to the core set, for example the target A in Figure 2, the front-end will choose a back-end node according to the designated request scheduling algorithm and modify the URI in the content of request packet to correspond with the path where the target core file is stored. If the requested web file belongs to the part set, the front-end would change the URI in the content of request packet to correspond with the path where the target part file is stored, and then check if this request packet needs to be handed off. If handoff is needed, then the front-end would handoff the request to the back-end which has the requested data. Lastly, if the requested web file neither belongs to the core set nor the part set, the front-end forwards the request to the back-end according to the assigned request scheduling algorithm.

3.2 Multiple TCP Rebuilding

The TCP Rebuilding technique [6] allows the front-end to transfer its TCP state of an established connection with a client to a back-end node. After the TCP state has been transferred, the chosen back-end could respond the request to the client directly, bypassing the front-end node.

We extend the TCP Rebuilding technique, called Multiple TCP Rebuilding, to efficiently support HTTP/1.1 persistent connection, as shown in Figure 3, by adding the functionality in the front-end to migrate a connection between the back-end nodes. Thus, the different requests in the same connection could be distributed to different back-end nodes in the presence of persistent connection.

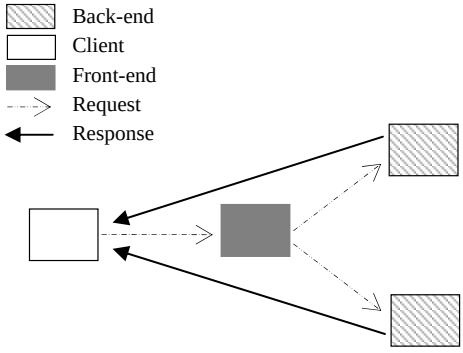


Figure 3: Multiple TCP Rebuilding

To allow Multiple TCP Rebuilding, the back-ends should be installed with the TCP-Rebuilding technique and the front-end should be customized to have functionalities as follows. First

of all, the front-end should do the content-aware distribution at the granularity of individual requests. Second, if the subsequent requests have been scheduled and then distributed to a different back-end node, the front-end should disconnect the previous connection. In order to increase the performance of migrating a connection, we implement the Multiple TCP Rebuilding to generate and send the RST packet for disconnection after handing off the request to the different back-end as shown in Figure 4.

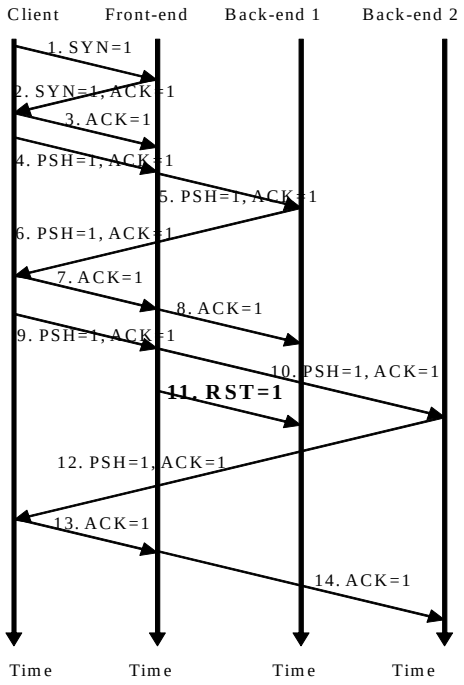


Figure 4: Sending RST packet after connection is migrated

The data flow of Figure 4 is described in the following:

1~3. The font-end performs the three-way handshaking with the client.

4. After receiving the request packet, the front-end will call the dispatcher module to select a back-end server based on the designated request scheduling algorithm and then forward the request to the chosen back-end.

5~6. When the selected back-end receives the request, it will rebuild the connection with the client using the TCP Rebuilding technique and then process the request and respond the data back to the client directly bypassing the front-end.

7~8. While the front-end receives the ACK packet from the client, it will forward the packet to the chosen back-end immediately.

9~11. When the front-end receives the subsequent request from the same connection, if it decides to distribute the request to another back-end, it then hands off the request to the back-end 2 using Multiple TCP Rebuilding technique. After the connection is rebuilt in the back-end 2, a RST packet is generated and sent to the back-end 1 to disconnect the no longer needed connection.In fact, the Multiple TCP Rebuilding is a high cost function including the overheads for the front-end to generate a RST packet to disconnect with the previous back-end server and modify the proper connection record, for the back-end 1 to tear down the connection when it receives the generated RST packet, for the back-end 2 to rebuild the connection with the client using the TCP Rebuilding technique. Because of these overheads, Multiple TCP Rebuilding should be used only when it is beneficial, in this paper, for improving cache hit ratio.

12~14. After the back-end 2 rebuilds the connection with the client, it would respond to the client directly and the subsequent packets would be forwarded to the back-end 2.

3.3 Dispatching Policies

Request scheduling algorithms implemented in LVS only estimate the loads of back-ends in the granularity of connection. In this section, we present two request scheduling algorithms that use more fine-grained method to measure the loads of back-ends to achieve better load sharing.

The idea of Weighted Least Request (WLR) scheduling algorithm comes from the Weighted Least Connection (WLC) algorithm, which is the most efficient one in the LVS [5]. We count the numbers of requests rather than the numbers of connections to measure the server's loads of back-ends more correctly. The WLR scheduling would select the target real server with the minimum value of the number of active requests divided by the server's weight.

The Weighted Least Traffic (WLT) scheduling algorithm measures the server load more fine-grainedly than WLR. The content-based dispatcher could examine the content of each request and then get the size of requested data from the URL table in the LVS-CWARD. Therefore, we could conjecture the server's load by the transferred data size of back-ends and select the least loaded server to service next connection. The WLT scheduling would select the target real server with the minimum value of the number of active transferred data size divided by the server' s weight.

4. Performance Evaluation

4.1 Experimental Environment

Our testbed consists of one front-end node, eight back-end nodes, and ten clients, connected to a single 24-port fast-ethernet switch. The environment is a stand-alone local area network with no disturbs from external network traffic. The packet forwarding mechanism is set to be Direct Routing [5]. The front-end node is with Intel P4 3.4G, which contains

DDR 256. The eight back-end nodes are with Intel P4 3.4G, each of which contains DDR 256/128. Each node runs Linux 2.4.18.

The benchmarks we used in this research are WebBench [23] and http_load [3]. WebBench is a licensed PC Magazine benchmark program developed by VeriTestthat to measure the performance of web servers. It has two components: a controller and clients. The controller controls clients for proposing requests, recording and summarizing the experimental data. It calculates two overall server scores: requests per second and throughput in bytes per second. The http_load is a freeware developed by ACME lab to test the throughput of a web server.

In order to perform trace-driven benchmarking, we modify it to replay the log in trace in order, instead of random. We use two publicly available traces from the Internet Traffic Archive [9], and they are traces of NASA Kennedy Space Center and ClarkNet web servers. The working sets used in this research are derived from these two logs.

4.2 Experimental Results

4.2.1 Overhead Evaluation of the Front-end

In this experiment, we setup the WebBench to repeatedly request the same web page of a given size, for the purpose to measure the overheads caused from our proposed system. The requested web pages are 1KB. The 1KB web page is chosen because the entire HTTP response could be transferred in a single Ethernet frame. We do not setup the LVS-CWARD to prefetch the target web page into RAM in this experiment, since our purpose is to investigate the additional overheads caused from this content-aware dispatching web system as compared with a content-blind web system. The additional overheads would include examining URL, looking up URL table, and modifying URL. In this experiment, the content-blind web cluster system, LVS, was used as the contrast. The request scheduling algorithm was WRR. Figure 5 shows the result.

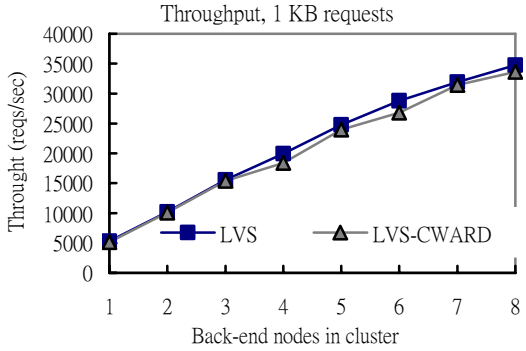
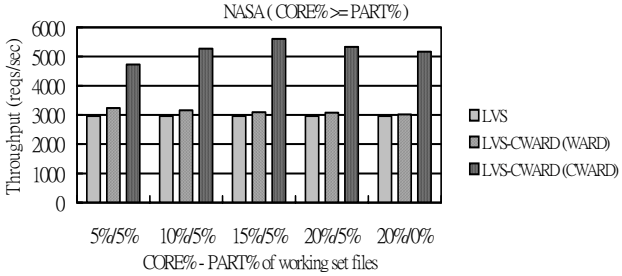


Figure 5: Throughput (reqs/sec), 1KB

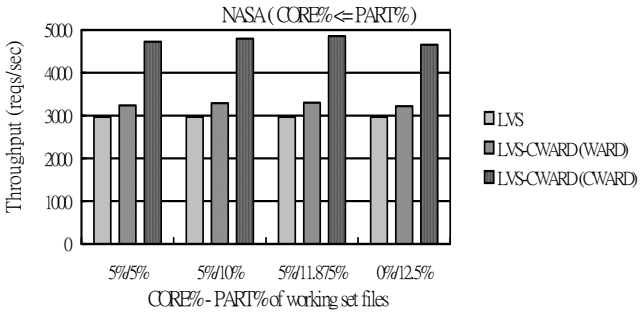
As shown in Figure 5, the throughput increases almost linearly with the size of the cluster. The overheads caused from our content-aware web cluster only degrade performance slightly. The throughput of LVS-CWARD is only 3.2% less than that of the LVS in the 1KB experiment.

4.2.2 Trace-driven Benchmarking

In this experiment, we evaluate our proposed system with two realistic workloads, and the workloads are derived from the logs of NASA and ClarkNet. The benchmark used in this experiment is http_load. In order to perform trace-driven benchmarking, we modify http_load to replay the log in order, instead of random. Besides, because there are ten clients in our testbed, so we split the log into ten parts in the Round-Robin manner, and each part for a client. The experimental web cluster consists of one front-end node, eight back-end nodes, and ten clients. The request dispatching scheduling algorithm used in this experiment is Weighted Round-Robin. Figures 6 and 7 shows the result.



(a) Trace-driven Benchmarking, NASA (CORE%>=PART%)



(b) Trace-driven Benchmarking, NASA (CORE%<=PART%)

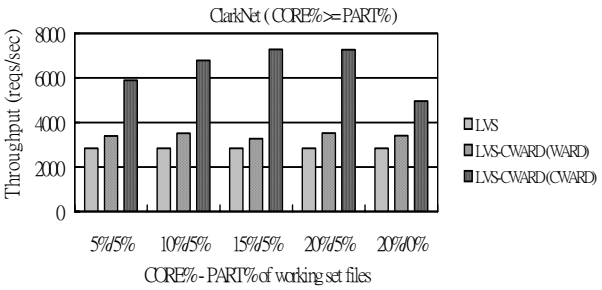
Figure 6: Trace-driven Benchmarking – NASA Trace

In Figures 6 and 7, the 5%/5% (core%/part%) means that the core set (i.e. the most frequently accessed files) has 5% of working set files, and the part set (i.e. the less frequently accessed files) in each node has 5% of working set files. Besides, LVS-CWARD (WARD) means that the LVS-CWARD platform adopts the WARD strategy, and LVS-CWARD (CWARD) means that the LVS-CWARD platform adopts the CWARD policy. The difference between LVS-CWARD (CWARD) and

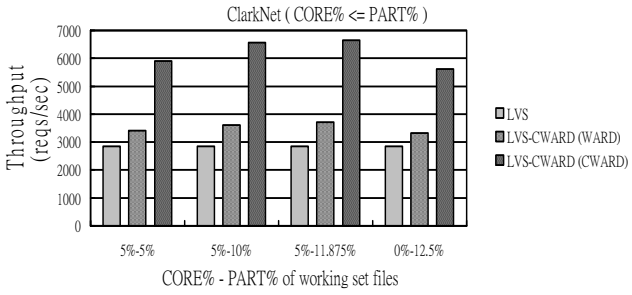
LVS-CWARD (WARD) is that the former has prefetched web pages into server's RAM, but the latter does not.

In this realistic workload, the advantage of our proposed LVS-CWARD is remarkable. The degree of locality of reference in NASA trace is quite high since when caching 10% of most frequently accessed files, it could achieve 96% cache hit ratio. In this experiment with 8 back-end nodes, we prefetch 45-100% working set into overall servers RAM. Thus, exceeding 96% of HTTP requested data would be accessed directly through server's RAM, bypassing disk I/O. Therefore, the performance gain of our proposed web cluster is obviously large. As shown in Figure 6, the performance of our proposed LVS-CWARD (CWARD) is 57-88% and 45-81% better than the LVS and LVS-CWARD (WARD), respectively.

Figure 7 shows that LVS-CWARD (CWARD) outperforms LVS and LVS-CWARD (WARD) by 107-156% and 69-122% respectively in the ClarkNet trace among different combinations of core and part sets.



(a) Trace-driven Benchmarking, ClarkNet (CORE%>=PART%)



(b) Trace-driven Benchmarking, ClarkNet (CORE%<=PART%)

Figure 7: Trace-driven Benchmarking – ClarkNet Trace

5. Conclusion

We have designed and implemented a high-performance and scalable content-based web cluster named LVS-CWARD. The goals of the LVS-CWARD is to improve cache hit ratio and load-sharing of the web cluster by taking content in requests and workloads into account in distributing requests.

Experimental results of practical implementation on Linux show that our proposed kernel-level content-based web switch is efficient and scales well without being the system bottleneck of the cluster. Moreover, the trace-driven benchmarking with the working sets derived from the logs of NASA and ClarkNet demonstrates that our proposed system can achieve up to 107% and 164% better performance than the layer-4 LVS web cluster respectively.

Further research is needed for optimizing the way of prefetching web pages in the server RAM. Besides, how to efficiently prefetch dynamic web contents into server RAM is another issue that needs to be investigated.

References

[1] Mohit Aron, Peter Druschel, and Willy Zwaenepoel, "Efficient Support for P-HTTP in Cluster-Based Web Servers," Annual USENIX Technical Conference, Monterey, CA, June 1999.

[2] Ludmila Cherkasova and Magnus Karlsson, "Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD," In Proceedings of the 3rd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, San Jose, CA, pp. 212-221, June, 2001.

[3] http_load, http://www.acme.com/software/http_load/.

[4] Ying-Dar Lin, Ping-Tsai Tsai, Po-Ching Lin, and Ching-Ming Tien, "Direct Web Switch Routing with State Migration, TCP Masquerade, and Cookie name Rewriting," Global Telecommunications Conference, 2003 (GLOBECOM '03) , Vol. 7, pp. 3663-3667, San Francisco, December 2003.

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

[6] H. H. Liu and Mei-Ling Chiang, "TCP Rebuilding for Content-aware Request Dispatching in Web Clusters," Journal of Internet Technology, Vol. 6, No. 2, pp. 231-240, April 2005.

[7] V. S. Pai, M. Aron, G Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers," Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October. 1998.

[8] Seon-Yeong Park, Dohyun Park, Joonwon Lee, and Jung Wan Cho, "Efficient Inter-backend Prefetch Algorithms in Cluster-based Web Servers," HPC Asia, September. 2001.

[9] The Internet Traffic Archive Website, http://ita.ee.lbl.gov/.

[10] WebBench Website, http://www.etestinglabs.com/benchmarks/webbench/webbench.asp.