Locality-Aware Request Distribution with Frequency-based Replication in Web Server Clusters

Shang-Yi Zhuang, Yu-Chen Yeh, Mei-Ling Chiang
Department of Information Management
National Chi-Nan University, Taiwan, R.O.C
Email: {s95213532, joanna, s97213527}@ncnu.edu.tw

Abstract

As services of WWW popularize and evolve, the capability of servers is highly concerned for conforming to modern and future requirements. The cluster-based web system is one of the solutions to meet the above requirements. Recently, in this cluster-based web system, content-aware request distribution policies become an important issue to effectively dispatch requests from clients to servers.

In this research, we propose a new dispatching policy named LARD/FR which improves from the well-known LARD/R policy. LARD/R policy allows a set of the back-end servers (server set) to serve the same target web object to increase the cache-hit ratio of back-end servers. By using a frequency-based mechanism, LARD/FR achieves higher cache hit rates of servers than LARD/R. We have implemented the proposed distribution policy on our LVS-CAD platform and practical experiments show it outperforms some other content-aware request distribution policies.

Keywords: Web Cluster, Content-aware Request Distribution.

1. Introduction

Due to the rapid growth of Internet activities, servers must be capable to handle heavy demand with the constraint of costs. There should be a scalable way to construct a set of cost-effective servers and service should be provided in a condition of 24x7 availability. Moreover, such a system should be manageable for administrators. Cluster-based web system is a solution to meet the above requirements. It consists of a front-end server and multiple back-end servers. A front-end server is also called web-switch that is responsible for distributing requests to the back-end servers. Back-end servers receive the packets from front-end server and actually handle those requests from clients.

When it comes to the issue of load balance in the cluster-based web system, there are two different platforms for this purpose. One is called content-aware platform [1] employing a layer-7 web switch, the other is called content-blind platform [1] employing a layer-4 web switch. Packet distribution of a layer-4 switch depends on the information of TCP/IP. In the other sides, packet distribution of a layer-7 switch depends on the information of HTTP content.

Some web clusters operates in content-blind platforms which employ a layer-4 web switch. However, it does not meet the requirements while the cluster needs to provide different quality of services or to dispatch requests based on request contents (i.e. URI). In contrast, a content-aware web cluster does. In such a cluster system, a layer-7 web switch distributes requests according to the request contents. Those contents can be important information for reaching better load balance among back-end servers. This difference from a layer-4 cluster allows system resources to be used more effectively.

There are some content-aware distribution policies which achieve load balance and high cache hit rates of servers, such as LARD [2], LARD/R [2], CAP [3], WARD [4], CWARD [5]. Nevertheless, when it comes to these issues, the above policies do not focus on the influence of access frequency of each web file. We believe that the frequency-based mechanism could utilize the whole system resources of a web cluster more effectively. In this research, we have proposed a new content-aware request distribution policy named LARD/FR and implemented it in the LVS-CAD web cluster [6]. This policy bases on the LARD/R policy while adopting a frequency-based mechanism to determine whether a back-end server should be added into a server set to serve specific requests which occur frequently.

The experimental results from practical implementation on Linux show that the proposed LARD/FR policy outperforms the well-known Weighted Round-Robin (WRR), LARD, and LARD/R policies.

2. Background and related work

This section introduces the LVS-CAD that uses TCP Rebuilding mechanism and fast handshaking mechanism to enable content-aware request dispatching. Finally, we introduce some content-based request distribution policies.

2.1. LVS-CAD web cluster

In our previous work [6], we have implemented a content-aware dispatching platform called LVS-CAD based on Linux Virtual Server (LVS) [7], which is a set of independent Linux-based servers acting as a single server to serve requests from clients.

In this platform, we apply TCP Rebuilding mechanism on each back-end server and fast handshaking mechanism on the front-end server. In the LVS-CAD platform, the front-end server can use not only content-aware dispatching policies, but also various content-blind dispatching policies of LVS. The IPVS-CAD module is modified from IPVS module in LVS system to apply fast handshaking mechanism. The fast handshaking mechanism can perform three-way handshaking with clients at IP layer instead of TCP layer so that it is more efficient than the ordinary three-way handshaking.

2.2. Content-based request distribution policies

There are several existent content-based request distribution policies such as locality-aware request distribution (LARD) [2], workload-aware request distribution (WARD) [4], content-based workload-aware request distribution (CWARD) [5], and the content-aware dispatching policy (CAP) [3].

The Locality-Aware Request Distribution (LARD) [2] is a content-based request distribution method which aims at simultaneously achieving load balancing and high cache hit rates in back-end servers while improving the performance of cluster-based web system.

LARD has two variant strategies to accomplish the goals: basic LARD and LARD with Replication (LARD/R) [2].The basic LARD always assigns one back-end server to serve the target data. In the beginning, while the front-end server gets the request, it chooses the least-loaded back-end server from all the back-end servers and then maps the request to this back-end server. If the target of the subsequent request has already been mapped to one of the back-end servers, the front-end server would forward the request to the back-end server which has served the target previously. In addiction, if the back-end server which serves the target is overloaded, the front-end server will selected another least-loaded back-end server to serve this request and the following identical requests will also be sent to the new selected back-end server.

Figure 1 illustrates the operation of LARD with the assumption of two back-end servers and three types of target web objects A, B, and C. The first back-end server has served the target A previously and the second back-end server has served the target B and C previously. Both back-end servers are not overload. The front-end server routes the requests for target A to the first back-end server which serves target A, whereas it routes the requests for targets B and C to the second back-end server which serves target B and C.

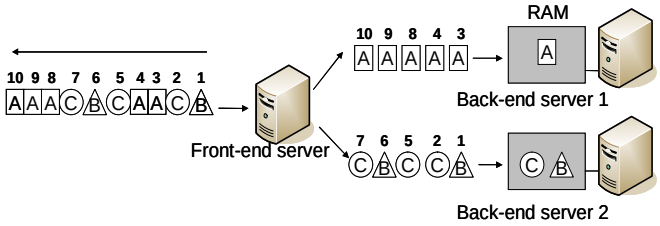

Figure 1. Locality-aware request distribution

The basic LARD only assigns a single back-end server to serve the target web object. This may result in the overload of some servers that serve frequently accessed web objects. In order to avoid the problem, the variant strategy, LARD with Replication (LARD/R), was developed. LARD/R allows a set of the back-end servers to form a server set to serve the same target web object. The front-end server maps the request to the back-end server which is the least-loaded one in the target server set.

Workload-Aware Request Distribution (WARD) tries to takes workload into account in dispatching requests. WARD identifies a small set of files called core which is requested most frequently, and the core file can be served by all back-end servers. The rest of files called part are equally partitioned and served by different back-end servers.

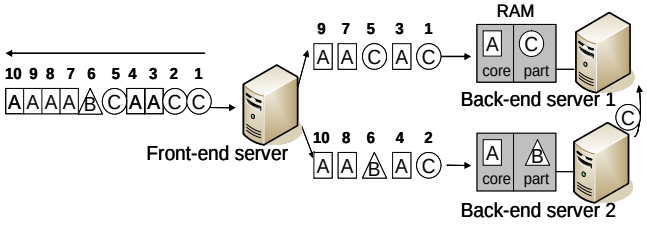

Figure 2. Workload-aware request distribution

As shown in Figure 2, if a client's request belongs to the core file (e.g. target A), any server can serve this request. However, if the request belongs to the part files (e.g. target B and C), it would be dispatched to the back-end server which is responsible for serving this requested file. If the subsequent request can not be served by the current server, TCP handoff is needed to

handoff the TCP connection to the proper server for handling the request. In Figure 3, we assume the target B is served by the back-end server 2 and the target C is served by the back-end server 1. When the back-end server 2 receives a target C request, it will handoff the TCP connection to the back-end server 1 which is responsible for serving the target C.

For a server cluster, performance would be poor if there are too many TCP handoffs. WARD can minimize the overhead by serving each request for core files locally on each cluster server. WARD also tries to minimize the overhead by maximizing usage of the servers' RAM to cache core files and a portion of partitioned files.

The goal of Content-based Workload-aware Request Distribution (CWARD) is to achieve higher cache hit rates in back-end servers and utilize system resource more effectively. CWARD adopts WARD like strategy and pre-fetches a small set of most frequently accessed files into servers' RAM to increase the performance of the whole web cluster. The method of pre-fetching files could decrease times of data transferring between RAM and disk when back-end servers handle requests.

The main goal of Client-Aware Policy (CAP) is to improve load sharing in web cluster providing multiple classes of services. CAP classifies requests into four classes, namely normal (N), CPU bound (CB), disk bound (DB), and disk and CPU bound (DCB) services. Then, round-robin policy is used to dispatch requests in each class to balance the load for each class of services in back-end servers. Therefore, this policy can equally distribute workload of each class among back-end servers.

Figure 3 illustrates an example of CAP policy. The front-end server dispatches requests to the back-end servers according to the four classes with the round-robin policy. The processing order is shown in each server from right to left.

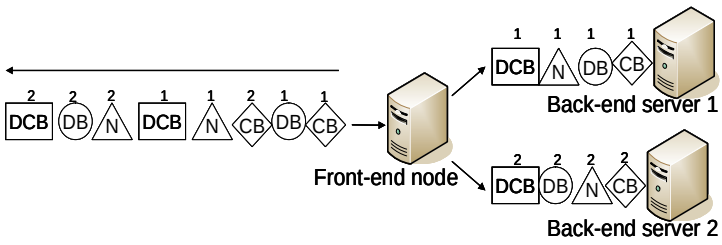

Figure 3. Client-aware policy

3. Proposed LARD with frequency-based replication policy

In this section, we present the design and implementation of LARD/FR policy. The conception of the policy is that more frequent the target data accesses, more servers work for it. The access frequency can be used to define hot web files as web files accessed frequently and cold web files as web files accessed infrequently. Keeping those hot file in RAM of back-end servers is important to optimize use of the whole web system.

For categorizing those hot files and cold files, the front-end server has to calculate the access frequency of each web file. Nevertheless, over complicated algorithms cause the overhead of the front-end server to slow the web system down.

To calculate the access frequency, we can set a timer to measure access time of each web file during a constant time, but the method wastes much power of front-end server on scanning all web files for calculating access frequency of each web file. Therefore, we replace access time of each web file with access time of identical requests. Access frequency of an identical request is calculated by analyzing how many identical requests have been received in the past period of time when the front-end server receives the request. By this way, the front-end server could easily know the access frequency of the identical request.

3.1. Design and implementation

The most important issue is to determine whether the requested web file has been frequently accessed in the past period of time. We devise the following equation in Figure 4 to determine whether the web file is frequently accessed.

$$\sum_{i=1}^{N} Ci \le \frac{TQ}{|ServerSet(R)|}(i = 1,2,...,N)$$

Figure 4. Illustration of LARD/FR threshold equation

In this equation, *TQ* means the time span to calculate the access frequency and the *Ci* is the time span between two identical requests' arrival time. *R* is the requested web file of a request and the *ServerSet(R)* is the number of back-end server which serves the web file *R*.

The scheme works by setting *N* to a constant before front-end server begins to run. Then, the front-end server set a variable to zero and use it for counting the number of times that identical requests occurs. When the variable reaches *N*, the front-end server checks the equation. If the equation is true, we can know that (*N*+1) identical requests have arrived within the time span *TQ/|ServerSet(R)|*. Therefore, more back-end server will serve a web file when the web file is more frequently been accessed.

An example of calculating the access frequency of a web file for a period of time is shown in Figure 5. Each node represents an arrival of a request. *Ti* is defined as a request's arrival time and *Ci* is the time span between *Ti* and *Ti+1*. *TQ* is the time quantum used to limit the time range for summing up requests' accessed times. We set *N* as the access threshold that determines whether this requested web file is frequently accessed and needs to add another back-end server into the server set for serving this request. When a request arrives, the front-end server will check whether the equation is true. If it is true, the front-end server will add another least-loaded back-end server to the server set of the request which means that this request has been received more than (*N*+1) times within the past period of time *TQ/|ServerSet(R)|*. If many identical requests have arrived in the span of time *TQ/|ServerSet(R)|*, it means the requested file is a hot file (i.e. frequently accessed web file). Then, the variables used to calculate the access frequency of the request will be reset, otherwise the equation will always be true and the front-end server will insert a back-end server into the server set of this request when a following identical request arrives. In Figure 5, we set *N* to 2. When the request T_3 has arrived, the equation becomes true which means the accessed web file is a frequently accessed file. The front-end server then adds one least-loaded back-end server to the request's server set and resets the related variables for calculating the access frequency.

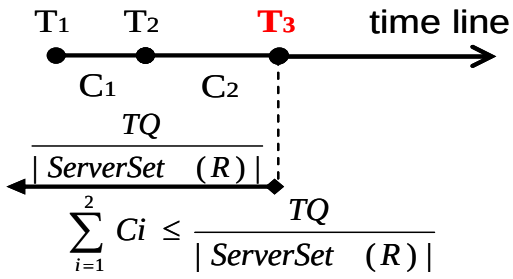

Figure 5. Illustration of LARD/FR policy with the condition of frequently accessed web file

Another example is illustrated in Figure 6. When the request T_3 has arrived, the equation is not true and the time quantum *TQ/|ServerSet(R)|* has expired. In this condition, it means the front-end server resets the variables for calculating the access frequency again, otherwise the equation will always be false.

In Figure 6, when the request T_3 has arrived, the equation is false and the time quantum *TQ/|ServerSet(R)|* has expired. This condition means the front-end server resets the variables for calculating the access frequency again, otherwise the equation will always be false.

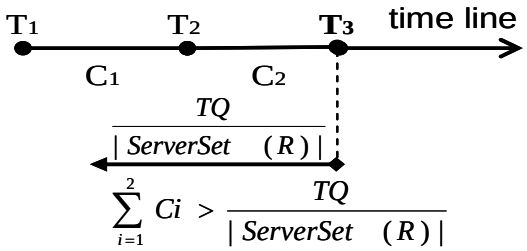

Figure 6. Illustration of LARD/FR policy with the condition of not frequently accessed web file

In our implementation of LARD/FR, we have to calculate the access frequency of each web file. As shown in the previous section, several *Ci* are summed up to determine if the Equation shown in Figure 4 is true. In our practical implementation, we can just subtract $T_{(n+1)}$ from T_1 to get the summation of *Ci*. Therefore, we do not record every time span of *Ci*. The front-end server only records the T_1 as the start time to calculate the access frequency when each request arrives.

```
fetch request r
odst ← current connected back-end server
// record the file's accessed number
increase the r's target accessed number by one
if (|serverset[r.target]| = 0) //have not been served before
   dst, serverset[r.target] ← least-loaded server
   set the r's starttime to now
else
   dst ← least-loaded server in serverset[r.target]
   if (time() - r's starttime <= TQ/|serverset[r.target]|) and (r's
         accessed number >= N)
      // frequently accessed web object
      dst , serverset[r.target] ← least-loaded server
      // reset variables
      reset the r's accessed number to zero and starttime to now
   if (time() - r's starttime > TQ/|serverset[r.target]|)
      // reset variables
      reset the r's accessed number to zero and starttime to now
      if (|serverset[r.target]| > 1)
         mdst ← most-loaded server inserverset[r.target]
         remove mdst from serverset[r.target]

  // consider the multi-handoff cost
   if (odst != dst) && (odst ∩ serverset[r.target])
      if (odst.activeconns - dst.activeconns <= P)
         dst ← odst

send r to dst
```

Figure 7. Pseudo code of LARD/FR

Figure 7 presents the pseudo code of LARD/FR. If the web file is the first time to be accessed, the front-end server would dispatch the request to the least-loaded back-end server and records the arrival time as the start time. When a web file has been accessed more than *N* times within a time quantum *TQ/|ServerSet(R)|*, the front-end server will insert a least-loaded back-end server to the server set.

In order to obtain the best performance, we also try to reduce the multi-handoff in our LARD/FR policy when it chooses one back-end server for serving the request within a persistent connection. The *P* is used to determine whether the front-end server has to handoff the request to a new back-end server.

In addition to add a back-end server into a server set, to remove a back-end server from a server set is also necessary. We simply remove the most-loaded back-end server from a server set if the request's server set contains more than one back-end server and time quantum *TQ/|ServerSet(R)|* has expired, which means this web file is not a frequently accessed file and the number of the back-end servers in the server set of this request should be reduced as well.

4. Experimental evaluation

In this section, we present performance evaluation of our proposed LARD/FR policy in Linux kernel 2.6.

In our experiments, web cluster includes eight back-end servers and one front-end server. Besides, ten computers are used as clients and each client runs httperf [8] benchmark to generate requests for testing the web cluster. All computers are connected by a ZyXEL Dimension GS-1124 switch. The hardware and software environment are shown in Table 1.

Table 1. Hardware and software environment

Item	Front-end	Back-end	Client
Processor	Intel P4 3.4G	Intel P4 3.4G	Intel P4 2.4G Intel P3 800MHz
Memory	DDR 512 MB	DDR 256 MB	DDR 256 MB
NIC (Mbps)	Intel Pro 100/1000	Intel Pro 100/1000	Reltek RTL8139 Intel Pro 100/1000 D-Link DGE- 530T
OS / kernel	Gentoo Linux/ 2.6.18	Gentoo Linux/ 2.6.18	Gentoo Linux/ 2.6.18
IPVS	1.0.4 / 1.21	X	X
Web Server	X	Apache 2.0.40 Apache 2.0.58	X
Benchmark	X	X	httperf
Number of PCs	1	8	10

We adopt trace log named WorldCup98 trace log [9] and used only six the requests on the day July 12, 1998 from AM 09:00 to PM 03:00. In this span of time, there were 1,974,360 requests accessed and totally 10,562 files accessed on clusters. The average file size is 161 Kbytes and the maximal file size is 2,824 Kbytes.

In the experiment of Figure 8, we compare the throughput of LARD/FR under different setting of *TQ* (time quantum) and *N* values with that of LARD, and LARD/R policies. There are ten clients generating requests under persistent connection, and each connection contains ten requests. As mentioned in Section 3, we have to set the *TQ* and *N* values in the Equation in Section 3.1. In this experiment, we set the *N* variable to 10. If a web file been accessed more than 10 times within *TQ* seconds, it is a frequently accessed web file. Then, we adjust the *TQ* variable to various values. Figure 8 shows that our LARD/FR outperforms LARD and LARD/R policies. The reason that LARD/FR policy performs better than LARD/R policy is because the LARD/FR policy uses frequency-based mechanism to achieve high cache rates of servers.

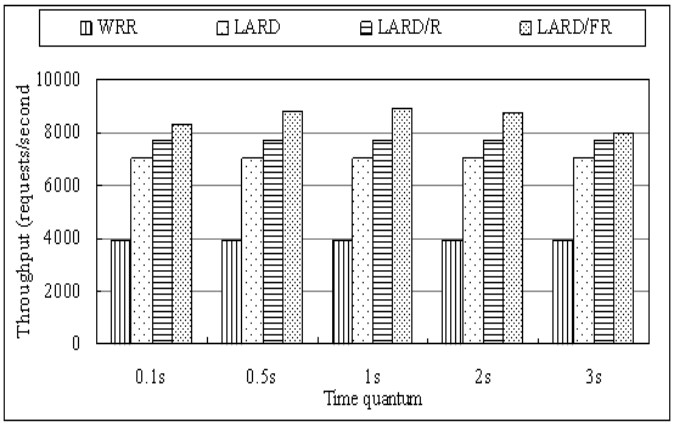

Figure 8. Performance of LARD/FR in variance parameter

Next we present the scalabilities of various request dispatching policies on cluster platforms. Figure 9 our proposed LARD/FR policy can obtain best throughput and outperforms LARD/R policy and WRR policy.

Finally, we want to know the maximal throughput the LARD/FR in LVS-CAD and Weighted Round-Robin (WRR) in LVS can achieve when the web clusters are under heavy load. Therefore, we adjust the request sending rate of httperf benchmark to send requests. The results are shown in Figures 10 and 11.

Experimental results demonstrate that the content-aware LARD/FR policy can achieve much higher throughput than the content-blind WRR policy. In our experiments, the throughput of LARD/FR policy can achieve up to 19239.9 requests per second, whereas the WRR policy can achieve up to only 5719 requests per

second. As a result, when the web cluster is under heavy load, the LARD/FR policy can handle almost four times of the requests per second than that of the WRR policy.

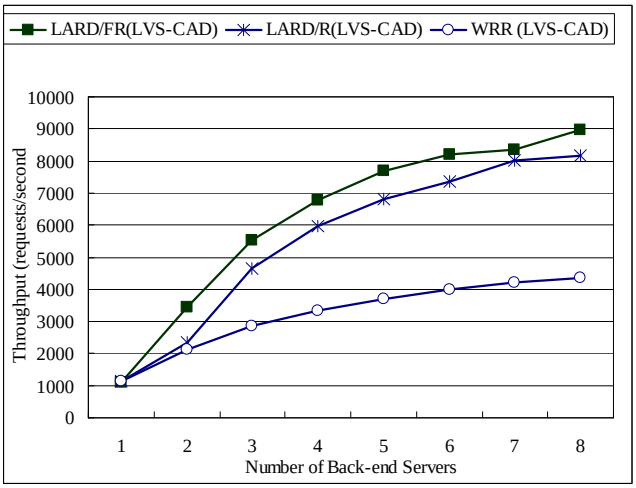

Figure 9. Scalability comparison of various policies and platforms

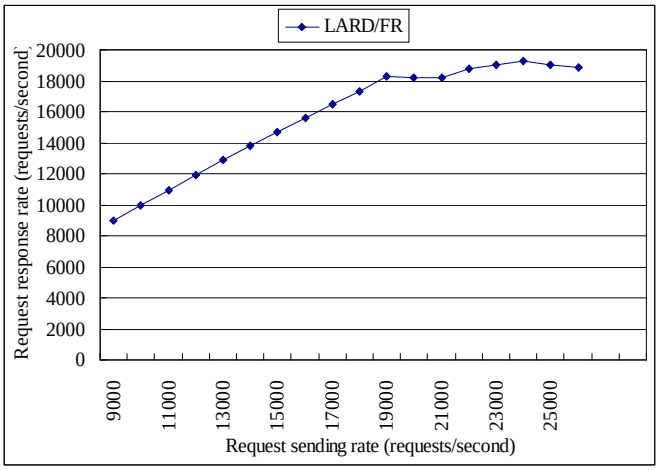

Figure 10. Maximal throughput of LARD/FR policy

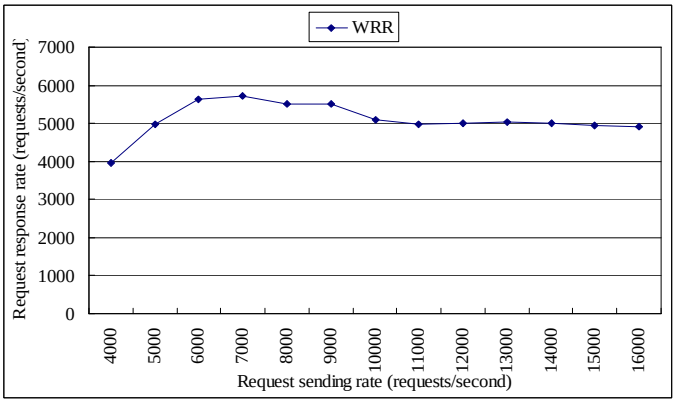

Figure 11. Maximal throughput of WRR policy

5. Conclusions

We have designed and implemented the frequency-based LARD/FR algorithm which improves the well-known content-aware LARD/R policy. In LARD/FR policy, frequently accessed web files will be served by more back-end servers. Our experimental results show that our proposed LARD/FR policy can get better throughput than the well-known WRR, LARD and LARD/R policies. Even when the web cluster is under heavy load condition, the proposed content-aware LARD/FR policy can achieve almost four times more throughput than the content-blind WRR policy.

6. References

[1] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," ACM Computing Surveys, Vol. 34, No. 2, pp. 263-311, June 2002.

[2] V. S. Pai, et al, "Locality-Aware Request Distribution in Cluster-based Network Servers," Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1998.

[3] E. Casalicchio and M. Colajanni, "A Client-Aware Dispatching Algorithm for Web Clusters Providing Multiple Services," Proc. of 10th Int'l World Wide Web Conference, Hong Kong, pp. 535-544, May 1-5, 2001.

[4] L. Cherkasova and M. Karlsson, "Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD," 3rd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, pp. 212-221, June 2001.

[5] M. L. Chiang, Y. C. Lin, and L. F. Guo, "Design and implementation of an efficient Web cluster with content-based request distribution and file caching," Journal of Systems and Software, Vol. 81, Issue 11, pp. 2044-2058, Nov. 2008.

[6] H. H. Liu, M. L. Chiang, and M.C. Wu, "Efficient Support for Content-Aware Request Distribution and Persistent Connection in Web Clusters," Software Practice & Experience, Vol. 37, Issue 11, pp. 1215-1241, Sep. 2007.

[7] Linux Virtual Server Website, http://www.linuxvirtualserver.org/, May 2007.

[8] The httperf, http://www.hpl.hp.com/research/linux/httperf/, 2007.

[9] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web site," Hewlett-Packard Technical Report HPL-1999-35R1, February 1999.