HyLARD: A Hybrid Locality-Aware Request Distribution Policy in Cluster-based Web Servers

Shang-Yi Zhuang, Mei-Ling Chiang
Department of Information Management
National Chi-Nan University
Taiwan, R.O.C
s95213532@ncnu.edu.tw
joanna@ncnu.edu.tw

Abstract

The cluster-based web server is cost-effective to be used in a popular web site. To balance the load of servers in a cluster and get better performance, many researches focus on the content-aware request distribution policies that take into account the content of requests in distributing requests from clients to servers.

In this paper, we have proposed a content-aware request distribution policy called Hybrid Locality-Aware Request Distribution policy (HyLARD). This policy mainly combines the advantages of the well-known locality-aware request distribution (LARD) policy and LARD with replication (LARD/R) policy, to improve the cache hit rates in servers' RAM while maximizing the utilization of servers' RAM. Besides, it also reduces the excessive multiple handoffs on persistent connections and improves load balancing on the server cluster. We have implemented it in the Linux Virtual Server with content-aware dispatching (LVS-CAD) Platform.

Keywords: web clusters, cluster-based systems, content-aware request distribution.

1. INTRODUCTION

Popular web sites often face the challenge to deal with huge amount of http requests at short time. A single server usually can't deal with such huge requests. Therefore, a cluster-based web server that can provide high scalability and high availability is often been used. Among the cluster-based servers, Linux Virtual Server (LVS) [6] which consists of several request-handling back-end servers and one request-dispatching front-end server is popularly used.

To get better performance, many content-aware platforms [2, 7, 10] have been developed to enable front-end server to apply sophisticated dispatching policies. The front-end server of a content-aware web cluster is also known as the layer-7 web switch in the OSI architecture. Many content-aware request distribution policies [3, 4, 5, 8] have been proposed to increase the throughput of the whole server cluster and balance the loads among the back-end servers, such as LARD [9], WARD [4], CWARD [5] and CAP [3] policies.

Locality-aware request distribution (LARD) policy is one of well-known content-aware request distribution policies. In LARD policy, front-end server dispatches the request of the same web object to the same back-end server. However, LARD may lead to load unbalancing due to the different popularity of web pages. Therefore, LARD/R policy was proposed to solve this problem by using server sets consisting of several back-end servers that may be selected to serve one specific request to prevent load unbalancing.

In this research, we propose the hybrid locality-aware request distribution policy (HyLARD) to enhance the LARD/R policy and implement it in the Linux Virtual Server with content-aware dispatching (LVS-CAD) Platform [7] which is based on TCP Rebuilding mechanism [7]. The LVS-CAD also supports multiple TCP Rebuilding like multiple handoff [7] which enables content-aware request distribution under persistent connections.

The rest of this paper is organized as follows. Section 2 introduces the background and related works. The design and implementation of our proposed Hybrid LARD (HyLARD) is presented in Section 3. Section 4 presents the experimental results. Finally, we conclude in Section 5.

2. BACKGROUND and RELATED WORKS

This section describes related content-aware request distribution policies. Section 2.1 briefly describes Linux Virtual Server. TCP rebuilding mechanism is introduced in Section 2.2. Section 2.3 presents the locality-aware requests distribution (LARD) policy and LARD/R policy. Workload-aware request distribution (WARD) is described in Section 2.4. Then, content-based workload-aware request distribution (CWARD) and content-aware dispatching policy (CAP) are described in Section 2.5 and Section 2.6 respectively.

2.1. Linux Virtual Server

Linux virtual server (LVS) developed by Wensong Zhang is a highly scalable and highly

available cluster-based server based on Linux. LVS consists of one front-end server and several back-end servers. There are three kinds of routing mechanisms on LVS: NAT (network address translation), IP tunneling and direct routing. Direct routing is the most efficient mechanism. The front-end server in LVS is also called layer-4 web switch according to the OSI layer. The layer-4 web switch dispatches requests according to the IP address and TCP port number in a packet.

2.2. TCP Rebuilding Mechanism

TCP rebuilding mechanism is used to enable the content-aware request distribution on LVS-CAD platform. In this platform, front-end server must perform three-way handshaking with clients to get the following requests containing HTTP content. Then, the front-end server can schedule the request to one of back-end servers according to the content of request. Therefore, TCP state must be migrated to the selected back-end server so that the selected back-end server can respond directly to the client.

TCP rebuilding is a light-weight TCP connection transfer mechanism which makes front-end server to transfer the established connection with client to a back-end server by using only the request packet.

2.3. Locality-Aware Request Distribution

The locality-aware request distribution (LARD) policy is a content-based request distribution policy which aims at achieving load balancing and high cache hit rates.

In LARD, when a request arrives, front-end server will choose the least loaded back-end server to serve the request and keep the mapping information of this request to the chosen back-end server. Then, the subsequently identical request will be served by the same back-end server to achieve high cache hit rates for reducing the disk I/O operations.

In LARD/R, when a request arrives, front-end will select one back-end server in the same manner as LARD does. Once the selected back-end server is overload, front-end will select another least loaded back-end server to serve this request. Thus, there are two or more back-end servers to serve this request and cache the document in RAM. These back-end servers form a server set of this request.

2.4. Workload-Aware Request Distribution

Workload-aware request distribution (WARD) is also a content-based request distribution policy which identifies a small set of most frequently accessed files called core to be served by all servers in the cluster. The rest of files called part are partitioned to be served by different back-end servers. WARD uses the ward-analysis algorithm to determine the file size of core files. Another idea in this policy is to make each back-end server could forward requests to other servers.

2.5. Content-Based Workload-Aware Request Distribution

Content-based workload-aware request distribution with core replication (CWARD/CR) aims to achieve high cache hit rates and reduce the data transferring between RAM and disks. It's similar to WARD policy which partitions web files into core and part. However, CWARD/CR allocates a small amount of RAM to pre-fetch core files into server RAMs that could effectively avoid core files to be replaced into disks due to cache replacement.

2.6. Content-Aware Dispatching Policy

The main goal of content-aware dispatching policy (CAP) is to improve load sharing in web cluster by classifying 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 is used in each class to balance the load in back-end servers. This policy can measure load of static and dynamic requests in back-end servers.

3. PROPOSED HYBRID LARD POLICY

In this section, we present the design and implementation of our proposed HyLARD policy.

3.1. Aiming For Reducing Swapping

We found that the LARD/R policy would not get a good performance when the web site contains many large-sized and frequently accessed files like WorldCup98. This may be caused by the situation when many large files are cached in most of back-end servers. If more large files are cached in many back-end servers, fewer amount of small files can be cached in back-end servers. That would increase disk swapping in back-end servers because of inefficiently using back-end servers' RAM. Besides, the throughput on the whole web cluster will decrease because disk I/O time is much longer than memory access time. In the other words, if too many back-end servers serve large files, the performance of the whole cluster will decrease. It means we should take into account the file sizes when front-end server choosing back-end servers to serve requests.

Therefore, we present a simple strategy to reduce the disk operation times. In HyLARD, front-end server distributes large files under LARD policy and distributes small files under LARD/R policy. So, a

large file will be served by only one back-end server no matter whether it is a frequently accessed file. This may lead to load unbalancing among back-end servers. However, it is a tradeoff between load balancing and reducing disk I/O.

3.2. Aiming with reducing multi-handoff

To support persistent connection in HTTP 1.1, a front-end server has to support multi-handoff such that it can hand off an established connection with client to another back-end server from original back-end server. LVS-CAD is based on TCP Rebuilding mechanism and also supports the multi-handoff mechanism.

Because content-aware policies need to analyze content of requests and route requests to the specific back-end server, every incoming request will be scheduled and dispatched. Contrarily, content-blind policies can get much better performance in persistent connection because a back-end server does not have to perform three-way handshaking operations in the same connection. Therefore, content-aware policies may get worse performance than content-blind policies in HTTP 1.1 environment.

With the aim for reducing multi-handoff, we take the cost of multi-handoff into account in front-end server when it chooses one back-end server for dispatching in a persistent connection. As shown in Figure 1, when a request has been scheduled and one back-end server has been selected under HyLARD algorithm, the front-end server needs to check whether it is worth to handoff the TCP connection from this ongoing connection.

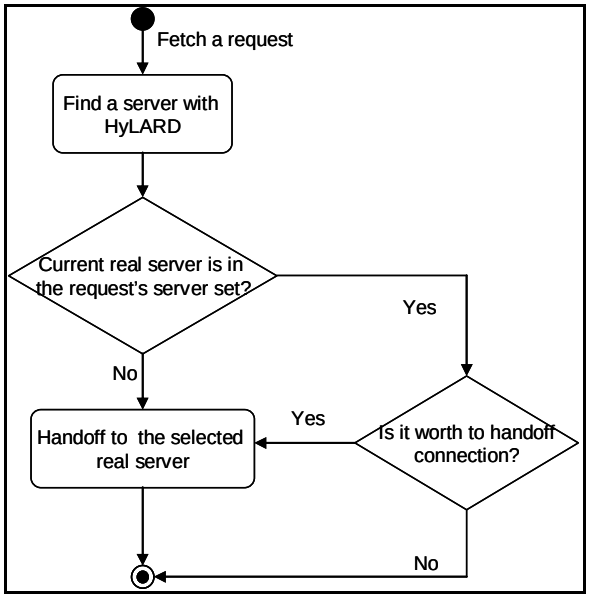

Figure 1 Reducing multiple handoffs in HyLARD algorithm

3.3. Implementation

Figure 2 shows the pseudo-code of HyLARD. The configurable variables such as *Threshold*, *M*, *P*, and *Q* appear in boldface. The *Threshold* in pseudo code is used to distinguish large files and small files. *M* is used to determine whether the least loaded back-end node in this server set is overloaded. Front-end server takes the value *P* of TCP handoff cost into account, after getting a back-end node through LARD/R policy. This request will be served by the original connected back-end server if the original back-end server is in the server set of this request and it is not worth to handoff request to another back-end server. Finally, when there is no back-end server to be added or removed from this server set for *Q* second, which means that this request is no longer the frequently accessed file. Then, one most loaded back-end server will be removed from this server set.

```
fetch request r
o ← previous back-end node
if target file size >= Threshold Kbytes
  // LARD
  if server[r.target] = NULL
    d, server[r.target] ← least load node
  else
    d ← server[r.target]
else
  // LARD/R
  l ← least loaded node
  if serverset[r.target] = NULL
    d, serverset[r.target] ← 1
else
  n ← least loaded node in serverset[r.target]
  if n != 1 && n.activeconns – l.activeconns >= M
    //overloaded
    d, serverset[r.target] ← 1
  if o != d && o ∩ serverset[r.target]
    //consider the multi-handoff cost
    if o.activeconns – d.activeconns <= P
      d ← o
  if serverset[r.target] > 1 &&
      time() – serverset[r.target].leastmod > Q
    m ← most loaded node in server[r.target]
    remove m from serverset[r.target]
if serverset[r.target] changed in this repetition
  serverset[r.target].lastmod ← time()
send r to d
```

Figure 2 Pseudo-code of HyLARD

4. PERFORMANCE EVALUATION

This section first presents our experimental environment. The experimental results of the HyLARD policy using WorldCup98 accessed log are presented. We compare the performance of HyLARD policy with two commonly used content-blind request dispatching policies, i.e. Weighted Round-Robin (WRR) [2,6] and Weighted Least-Connection (WLC) [2,6], and two content-aware request dispatching policies, i.e. Locality-Aware Request Distribution (LARD) and LARD with Replication (LARD/R).

4.1. Experimental Environment

Eight back-end servers and one front-end server are used in our web cluster. Ten clients run httperf [11] benchmark to generate requests and send requests to the whole web cluster. All computers connected to a D-Link DES-3225G switch. The hardware and software environment is shown in Table 1. Our proposed HyLARD, LARD, and LARD/R run on the LVS-CAD platform which supports content-aware dispatching and uses modified direct routing mechanism to route requests. The WRR and WLC run on the original Linux Virtual Server which supports only content-blind request dispatching.

TABLE 1 Hardware and software environment

Item	Front-end	Back-end	Client
Processor(MHz)	Intel P4 3.4G	Intel P4 3.4G	Intel P4 2.4G
Memory (MB)	DDR 512	DDR 256	DDR 256
NIC (Mbps)	Intel Pro 100/1000	Intel Pro 100/1000	Reltek RTL8139
OS	Red Hat Linux 8.0	Red Hat Linux 8.0	Red Hat Linux 8.0
Kernel	2.4.18	2.4.18	2.4.18-14
IPVS	1.0.4	X	X
Web Server	X	Apache 2.0.40	X
Benchmark	X	X	httperf
Number of PCs	1	8	10

4.2. Access Log

We use the publicly obtainable trace log named WorldCup98 trace log from the Internet Traffic Archive [12]. Because the whole trace log is extremely large, we used only six hours of 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 on clusters.

4.3. Experimental Setting

In our experiments, first we have to determine the threshold of file size to get the best performance. Figure 3 shows the performance of HyLARD in different “threshold” value. If threshold is set to 0, all requests are scheduled in pure LARD policy. If the threshold is set to high value which is higher than the maximal file’s size, HyLARD executes by always using LARD/R policy. Therefore, in order to take the advantage of LARD and LARD/R, it is necessary to set a proper threshold to get the best performance.

As showed in figure 3, the HyLARD policy gets the best performance when the threshold is set to 500. Therefore, in the following experiments, threshold is set to 500 in HyLARD policy. As threshold is set to 500, the number of large files occupies 16% of total web files. But, the total size of these large files occupies 83.02% of total web files size.

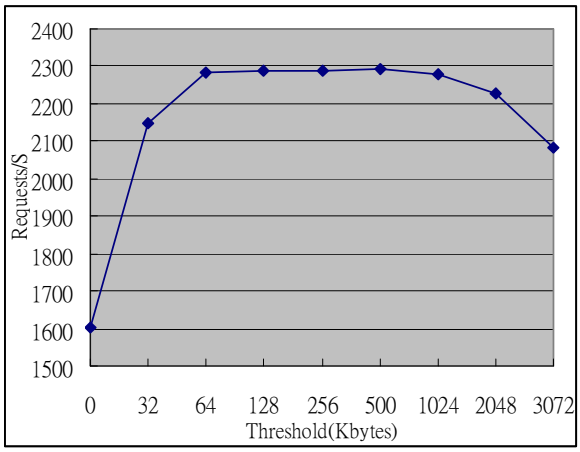

Figure 3 Performances of HyLARD

Other configurations are shown as follows. *M* is set to 2, which means the least loaded node of this server set is overload if it has two more active connections than the least loaded server. *P* is set to 1, which represents the TCP handoff cost. Finally, when the server set of a request has not been modified for *Q* (set to 60) seconds, the most loaded back-end server in the server set will be removed.

4.4. Performance Impact On Considering Multiple Handoffs

At first, we should confirm if the multiple handoffs will affect performance of web cluster. In this experiment, each connection sends ten requests so that front-end server would do multiple handoffs. Table 2 compares the performance between HyLARD without considering excessive TCP handoff and the enhanced HyLARD that considers reducing excessive TCP handoffs.

#test	Algorithm	requests/s
1	The original HyLARD without considering multi-handoff	2350.3
2	**The enhanced HyLARD**	**2439.1**

Table 2. Evaluate the performance of considering excessive multiple handoffs

The result shows that the original HyLARD without considering reducing excess multi-handoffs really gets bad performance compared with the enhanced HyLARD. Our proposed algorithm clearly reduces some multiple handoff overhands.

4.5. Performance Comparisons

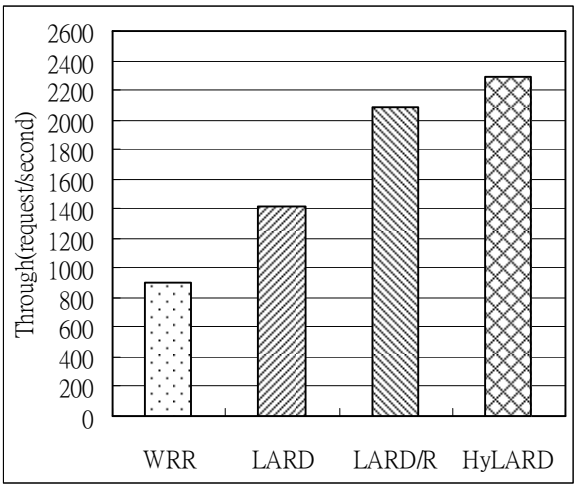

Figure 4 Comparison of various policies with no persistent connection

Figure 4 shows the first experimental result under the situation of only one request at each connection. In this setting, HyLARD can get better throughput than WRR since HyLARD can reduce the disk I/O and achieve high cache hit rates in the back-end servers. Besides, HyLARD also gets better throughput than LARD and LARD/R. As presented in Figure 4, HyLARD can get 153.74% better throughput than WRR, and outperform LARD and LARD/R by 61.47% and 10.11%, respectively.

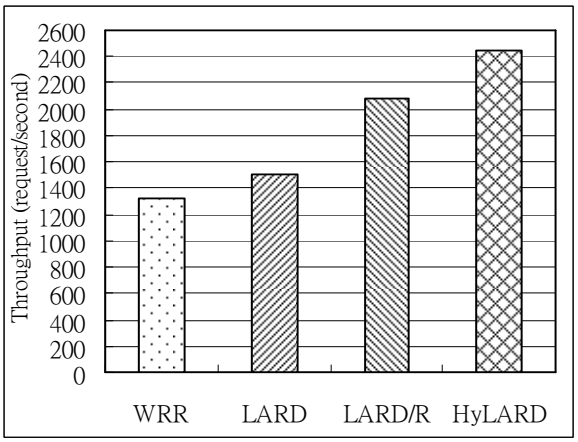

Figure 5 Comparison of various policies with ten requests per connection

Figure 5 shows the second experimental result under the situation of ten requests per connection. Because of the advantage of persistent connection, WRR can obtain better throughput than the result shown in the first experiment. The content-aware policies can not get as much benefit from persistent connection as WRR does since front-end server has to run dispatching policy at per coming request instead of per connection. Besides, excessive multi-handoffs would reduce the throughput of content-aware dispatching policies. However, HyLARD still performs 84.51% better throughput than WRR and 16.90% better throughput than LARD/R.

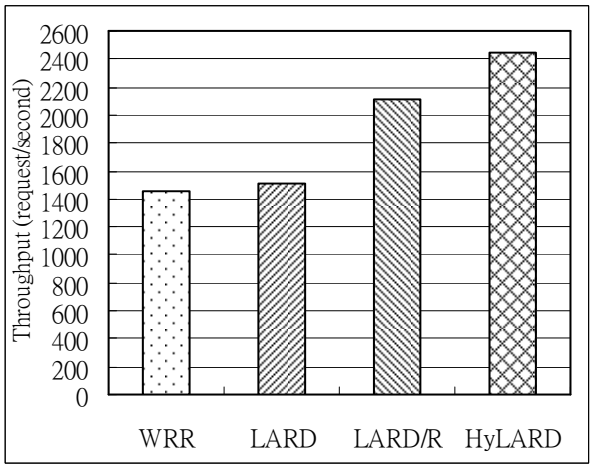

Figure 6 Comparison of various policies with twenty requests per connection

Figure 6 shows the third experimental result of comparing four policies under the situation of twenty requests per connection. The throughput of content-blind WRR policy gets better and is similar to LARD policy. Whereas, content-aware policies get only slight improvement. Since both HyLARD and LARD/R get small amount of improvement, the throughput of HyLARD is better than that of LARD/R about 16.67% which is similar to the result in the second experiment.

4.6. Performance Growth Under HTTP/1.1

In this experiment, we compare the performance growth between content-blind WLC algorithm and our proposed content-aware HyLARD algorithm.

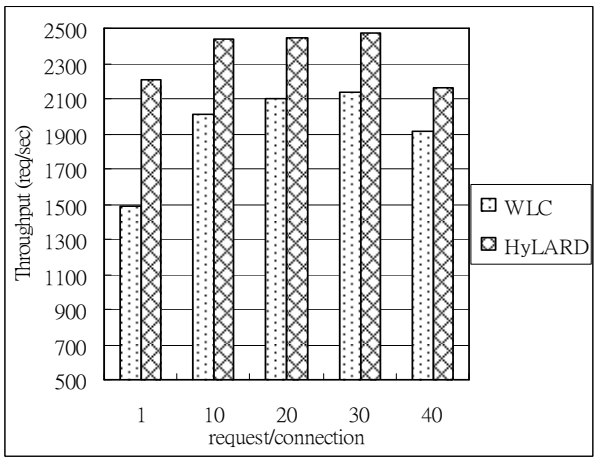

Figure 7 Performance growth in WLC and HyLARD

Figure 7 shows the performance growth in WLC and HyLARD when the number of requests per connection is increased. Although, content-aware policies have the multiple handoff overhands, our

enhanced HyLARD still can obtain much improvement in ten requests per connection compare with that in one request per connection. This is because we try to reduce the excessive multi-handoffs in our policy. Besides, the result shows the performance growth rate of WLC is larger than that of HyLARD in the first three throughputs because the content-blind policies do not have to hand off connections among back-end servers.

We can find the performance of content-aware HyLARD algorithm under ten requests per connection is almost same as that of twenty or thirty requests per connection. That is because the front-end handoffs requests under persistent connections and web files have the same locality under these environments.

However, the throughputs of WLC and HyLARD certainly decrease when each connection dispatches forty requests. The performance degradation may be caused by the load unbalancing among back-end servers.

To sum up, our proposed HyLARD content-aware dispatching policy can still get better performance than the well-known WLC content-blind request distribution policy even in the persistent connection environment. HyLARD also performs best among the content-aware request distribution policies.

5. CONCLUSIONS

In this paper, we have proposed an effective content-aware dispatching policy called HyLARD which combines the advantages of LARD and LARD/R policies. This research focuses on reducing disk I/O and balancing the loads among back-end servers to get better performance. We have implemented this policy on our previous work of LVS-CAD platform which is modified from LVS and could effectively support content-aware request distribution.

Our experimental results show that content-aware policies in the persistent connections environment would not increase as much throughput as the content-blind policies. This is because front-end server has to schedule each incoming requests and may need to hand off connections to different back-end servers. However, the cost of TCP handoff has been considered and reduced in HyLARD policy. Besides, the experimental results also show that our HyLARD policy can still obtain more throughput than WRR and LARD/R policies under HTTP/1.1.

The best threshold for deciding between LARD and LARD/R is found to be 500KB in Section 4. This threshold value depends on the amount of memory in the system, the number of nodes, the performance of each node, the performance of the disks, etc. i.e. it depends on the testbed. But, no matter what threshold value is set, the performances of HyLARD are better than that of LARD or LARD/R.

In this HyLARD policy, those controlled variables used need to be further studied, such as determining threshold between large files and small files on different workload, the cost of multi-handoff, and the span of time to remove one back-end server from a server set.

6. References

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

[2] 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.

[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, San Jose, CA, pp. 212-221, June 2001.

[5] Y. C. Lin, M. L. Chiang, and L. F. Gu, “System Support for Workload-aware Content-based Request Distribution in Web Clusters,” Journal of Internet Technology, Vol. 7, No. 3, 2006.

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

[7] H. H. Liu, M. L. Chiang, and M.C. Wu, "Efficient Support for Content-Aware Request Distribution and Persistent Connection in Web Clusters," to appear in Software Practice & Experience, 2006.

[8] M. Y. Lou, C. S. Yang, and C. W. Tseng, “Analysis and Improvement of Content-Aware Routing Mechanisms,” IEICE Transactions on Communications, Vol.E88-B No.1 p.227-238, January 2005.

[9] 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, San Jose, CA, Oct. 1998.

[10] M. D. Santo, N. Ranaldo, E. Zimeo, “Kernel implementations of Locality-Aware Dispatching Techniques for Web Server Clusters,” Proceedings of the IEEE International Conference on Cluster Computing, Dec. 2003.

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

[12] The Internet Traffic Archive Website, http://ita.ee.lbl.gov/, May 2007.