Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Design and implementation of an efficient web cluster with content-based request distribution and file caching

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

Department of Information Management, National Chi-Nan University, Puli, Taiwan, ROC

ARTICLE INFO

Article history:
Received 8 September 2005
Received in revised form 27 February 2008
Accepted 27 February 2008
Available online 14 March 2008

Keywords:
Content-based request distribution
Web cluster
Content-aware web switch
Content-blind web switch
Persistent connection

ABSTRACT

We have implemented an efficient and scalable web cluster named LVS-CAD/FC (i.e. LVS with Content-Aware Dispatching and File Caching). In LVS-CAD/FC, a kernel-level one-way content-aware web switch based on TCP Rebuilding is implemented to examine and distribute the HTTP requests from clients to web servers, and the fast Multiple TCP Rebuilding is implemented to efficiently support persistent connection. Besides, a file-based web cache stores a small set of the most frequently accessed web files in server RAM to reduce disk I/Os and a light-weight redirect method is developed to efficiently redirect requests to this cache. In this paper, we have further proposed new policies related to content-based workload-aware request distribution, in which the web switch considers the content of requests and workload characterization in request dispatching. In particular, web files with more access frequencies would be duplicated in more servers' file-based caches, such that hot web files can be served by more servers. Our goals are to improve cluster performance by obtaining better memory utilization and increasing the cache hit rates while achieving load balancing among servers. Experimental results of practical implementation on Linux show that LVS-CAD/FC is efficient and scales well. Besides, LVS-CAD/FC with the proposed policies can achieve 66.89% better performance than the Linux Virtual Server with a content-blind web switch.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

To deal with the explosive growth of the World Wide Web, the cluster-based web systems have been widely adopted to handle large amount of HTTP request efficiently. A cluster-based web server system (briefly, web cluster) consists of multiple web servers connected by a high-speed LAN. It employs a web switch also called front-end which distributes requests from clients among the request-handling servers also called back-ends to achieve load sharing and scalability.

The front-end of a web cluster could be classified as a layer-4 or layer-7 web switch according to the OSI layer at which the distribution decision is made (Cardellini et al., 2002). The layer-4 web switch dispatches the requests according to the IP address and TCP port, whereas the layer-7 web switch can perform content-aware request dispatching, dispatching the requests in accordance with the content (i.e. URI) examined from the requested packets.

In recent years, many studies (Andreolini et al., 2003; Aron et al., 1999, 2000a,b; Cardellini et al., 2002; Casalicchio and Colajanni, 2001; Cherkasova and Karlsson, 2001; Lin et al., 2003; Liu et al., 2007; Pai et al., 1998; Park et al., 2001; Sit et al., 2004; Wang, 2004; Yang and Luo, 1999; Zhang et al., 1999a,b) have focused on content-aware dispatching. Their works all demonstrate that using the content of requests and loading information, more intelligent request distribution algorithms such as improving disk cache hit rates can be developed. As a result, web clusters are more efficient in handling all types of requests. Moreover, with content-aware dispatching, partitioning web contents, building specialized web services among web servers, or maintaining session integrity can be achieved.

However, to be able to perform content-aware dispatching, the web switch should implement some mechanism such as TCP Splicing (Maliz and Bhagwat, 1998) or TCP Handoff (Pai et al., 1998). In our previous study, we also proposed the TCP Rebuilding (Liu et al., 2007), a light-weight TCP connection transfer mechanism that enables a web cluster to be content-aware. Since the HTTP request (i.e. GET URL) is sent after the TCP connection is established, the front-end must first establish a connection with the client before any distribution decision can be made. The TCP Rebuilding allows the front-end to transfer an established connection with a client to a chosen back-end by rebuilding the TCP connection in the back-end. After the TCP connection has been rebuilt, the chosen back-end could respond to the request of the client directly, bypassing the front-end. In particular, TCP Rebuilding could rebuild the TCP connection at one back-end using only the original HTTP request packet and no extra packets for connection transfer are required. Therefore, TCP Rebuilding is adopted to construct the content-aware platform in this research.

* Corresponding author.
E-mail addresses: joanna@ncnu.edu.tw (M.-L. Chiang), s1213526@ncnu.edu.tw (Y.-C. Lin), s3213534@ncnu.edu.tw (L.-F. Guo).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.02.069

Basing on our prior work (Lin et al., 2005), we have implemented a high-performance and scalable web cluster named LVS-CAD/FC (i.e. LVS with Content-Aware Dispatching and File Caching). The LVS-CAD/FC cluster implements a kernel-level one-way layer-7 web switch based on the TCP Rebuilding mechanism to perform content-aware dispatching and implements a file-based web cache that uses a small amount of memory dedicated to cache a small set of most frequently accessed files in servers. A light-weight method is then developed to efficiently redirect requests from clients to this file-based cache. To efficiently support HTTP/1.1 (Fielding et al., 1999) persistent connection, the fast Multiple TCP Rebuilding is implemented.

In this paper, we further propose new content-based workload-aware request distribution policies, which take into account the content in requests and workload characterization in distributing requests. Basically, a specific web request can be served by one of a certain amount of servers according to the designated request scheduling algorithm, whereas, the server set may be different for each request. The servers' file-based caches are organized in specific ways based on the workload characterization according to the proposed policies. Especially, frequently accessed web pages can be served by more web servers. Our main goals are to obtain better memory utilization and increase the cache hit rates in web servers while achieving load balancing. Consequently, disk swapping times and disk I/O times can be reduced, and cluster performance can be improved as well.

Experimental results of practical implementation on Linux show that LVS-CAD/FC is efficient and scales well. Moreover, performance evaluation with real-world traces (Internet, 2000) demonstrates that LVS-CAD/FC with the proposed request distribution policies can achieve 66.89% better performance than the layer-4 LVS web cluster (Linux, 2006).

The rest of this paper is organized as follows. Section 2 introduces briefly the background technologies and related works. The design and implementation of our LVS-CAD/FC is described in Section 3. The proposed content-based workload-aware request distribution policies are presented in Section 4. Section 5 presents the experimental results. We conclude in Section 6.

2. Background and related works

This section begins with the introduction of the content-blind dispatching platform – Linux Virtual Server. We then introduce existent mechanisms for content-aware request distribution in Section 2.2. Section 2.3 describes the existent mechanisms for dealing with HTTP/1.1 persistent connection in web clusters. Existent content-aware request distribution policies are discussed in Section 2.4. Finally, Section 2.5 describes other related works.

2.1. The content-blind dispatching platform – Linux Virtual Server

The Linux Virtual Server (LVS) (Linux, 2006; Zhang et al., 1999a,b) is a highly scalable and available server built on a cluster of servers. The architecture of LVS comprises a front-end server (FE) and several back-end real servers (BEs). The FE is a load balancer responsible for dispatching and routing requests from clients to the real servers. The BEs handle requests and respond to clients. The cluster system is transparent to clients as a virtual service using a single IP address.

LVS supports three routing mechanisms, namely Network Address Translation (NAT), IP tunneling, and direct routing (Mack, 2003). Among them, the most efficient mechanism is direct routing. Fig. 1 shows the packet forwarding flow of LVS using direct routing mechanism. The steps are described as follows. A client first sends a SYN packet to initiate a connection (Step 1). When the FE receives the SYN packet, it will start the scheduling module to select a BE, the SYN packet is then forwarded to the BE (Step 2). When the selected BE receives the SYN packet, it responds to the client with a SYN-ACK packet (Step 3). When the client receives the SYN-ACK packet, it responds with an ACK packet to establish a connection (Step 4). When the FE receives the ACK packet, it then forwards the ACK packet to the previously selected BE. When the BE receives the ACK packet, it changes its TCP state to the established state. The three-way handshaking is now completed (Step 5). The client then sends the HTTP request to the FE. The FE then forwards this HTTP request to the previously selected BE, and this BE then responds with an ACK packet to the client (Steps 6–8). The BE handles the request and forwards the requested data directly to the client (Step 9). The client responds to the FE with an ACK packet and the FE forwards the ACK packet to the previously chosen BE (Steps 10 and 11).

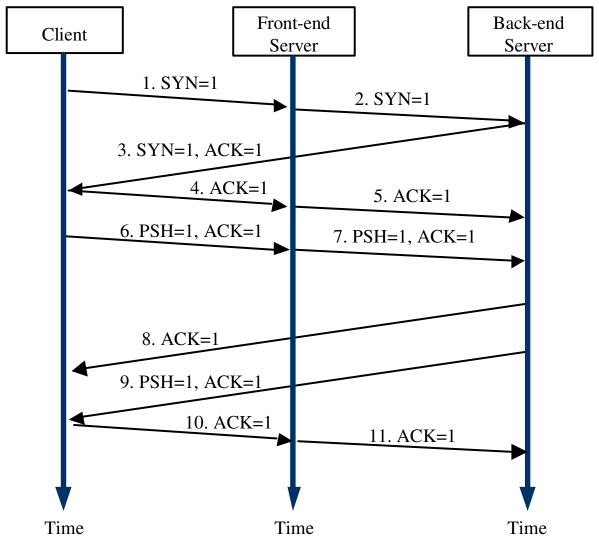


Fig. 1. Packet forwarding flow of LVS.

Since the BE responsible for serving requests in one connection is selected only when the FE receives a SYN packet for connection establishment, so the FE in LVS is a content-blind load distributor that cannot perform content-aware distribution.

2.2. Existent mechanisms for layer-7 web switch

In order for the front-end server to provide content-aware distribution, the request scheduling timing must be delayed until the request packet containing the HTTP content is received. Therefore, the front-end server must conduct TCP three-way handshaking with clients. After receiving a HTTP request from a client, the front-end server will schedule the packet and then transfer it to the selected back-end server.

2.2.1. Two-way mechanisms for layer-7 web switch

TCP Gateway, TCP Splicing (Maliz and Bhagwat, 1998), and Redirect Flows (Steven et al., 1999) belong to two-way mechanism for content-aware request distribution. In the two-way architecture, the response packets from back-end servers have to pass through the front-end server, which might limit the performance of the web cluster.

TCP Gateway should maintain two TCP connections for serving requests, since a client must first establish a TCP connection with the front-end server for sending out requests and the front-end server must establish another TCP connection with the chosen back-end server to process the request from the client. Besides, the front-end server processes the requests in the application layer, which would incur a high cost for copying data between the kernel and user space. In order to prevent such data copying, TCP Splicing is designed to splice two TCP connections in the TCP layer, such that all data are processed in the kernel and bypass the application layer. However, the front-end server still has to maintain two TCP connections in the same way as the TCP Gateway does. Moreover, a two-way mechanism might cause the front-end server to become the system bottleneck.

Redirect Flows is based on the NAT architecture. Through the NAT mechanism (Mack, 2003), they need not maintain two TCP connections. However, NAT is still a two-way mechanism and the performance of the web cluster is limited by the NAT architecture.

2.2.2. One-way mechanisms for layer-7 web switch

TCP Handoff (Pai et al., 1998), One Packet TCP State Migration (Lin et al., 2003), and TCP Rebuilding (Liu et al., 2007) belong to one-way mechanism for content-aware request distribution. In the one-way architecture, the response packets from back-end servers can be sent directly to the clients, bypassing the front-end server. This prevents the front-end server from becoming the system bottleneck. However, the processing is more complicated since in order for the back-end servers to respond directly to the clients, the TCP state of the front-end must be migrated to back-end servers, such that the back-end could send the requested web page directly to the client, bypassing the front-end. Therefore, one-way mechanism is more scalable than two-way mechanism.

TCP handoff works as follows. A client first initiates a connection with the front-end. When the front-end receives the HTTP request from the client, it migrates the TCP connection to the chosen back-end using its own proprietary protocol. After the connection is migrated, the back-end could process the HTTP request and forward the data directly to the client. In order to transfer TCP connection, besides the request packet itself, TCP handoff still needs extra packet or state information for transferring state during every handoff.

Instead of using custom protocol to transfer TCP connection state, One Packet TCP State Migration uses the TCP information in the original request packet to reconstruct the connection in the back-end. This mechanism implements a packet filter in each back-end server as the coordinator that maintains the TCP connection between the TCP module of back-end server and client. Besides, this mechanism needs to spoof packets between the packet filter and the TCP module on the back-end servers for three-way handshake, and the packet filter has to process and modify every inbound/outbound packets. These factors could affect the efficiency of back-end servers.

TCP Rebuilding is an efficient technique proposed in our previous study; it also uses the TCP information in the original request packet to rebuild the connection in the back-end. Especially, it does not require overheads incurred by extra packets and processing of packet filter for state transfer.

Fig. 2 (Liu et al., 2007) shows the packet forwarding steps. First, 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 (Steps 1–4). When the back-end receives the HTTP request, it rebuilds the TCP connection with the client using the TCP Rebuilding technique. In particular, the TCP connection is rebuilt without the packet filter needed in the One Packet TCP State Migration technique (Step 5). After the connection has been rebuilt, the back-end could forward the requested data to the client directly (Steps 6 and 7). When the front-end receives the ACK packet sent from the same client, it forwards the packet to the same back-end (Steps 8 and 9).

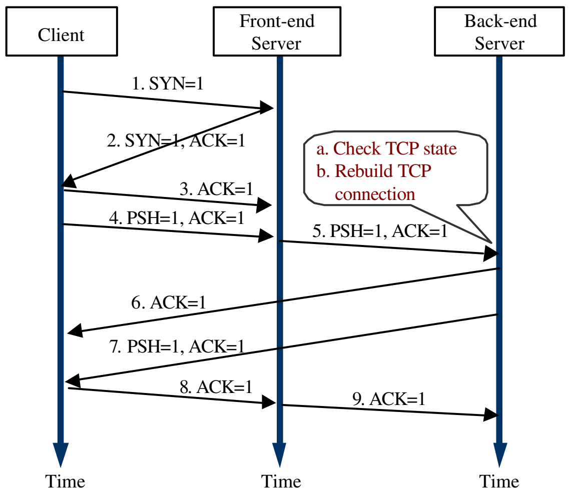


Fig. 2. Packet forwarding flow of TCP Rebuilding.

2.3. Existent mechanisms for handling HTTP/1.1 persistent connection

HTTP/1.1 persistent connection allows a client to send multiple requests in a single connection to reduce the overheads due to setting up and tearing down a TCP connection. However, it poses a problem for the web cluster that employs the content-aware request distribution. The problem is that the multiple requests in one connection may be scheduled to different back-ends. For example, the front-end schedules request 1 to back-end 1, and has connected to back-end 1. If the front-end schedules request 2 in the same connection to back-end 2, back-end 2 cannot process request 2 immediately because back-end 2 does not have the established TCP connection state. Therefore, the front-end should migrate the TCP connection to back-end 2, and then disconnect it with back-end 1.

Furthermore, if the web cluster partitions the web content among the various back-ends, then certain requests could only be served by certain back-end servers. In other words, one back-end server could not serve all types of requests. When HTTP/1.1 persistent connection is applied, if all the requests in the same connection are sent to the same back-end server, this would cause a serious “404 object not found” error since a back-end could not serve all the requests in a connection.

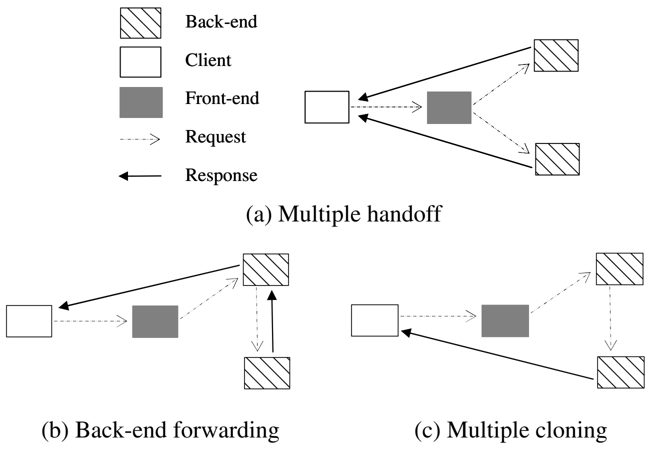


Fig. 3. Mechanisms for handling HTTP/1.1 persistent connection.

Aron et al. (1999) proposed the Multiple TCP Connection Handoff technique to deal with the HTTP/1.1 persistent connection problem. As shown in Fig. 3a, Multiple Handoff allows the front-end to migrate a connection between back-ends. Therefore, the front-end could distribute the requests at the granularity of requests. Like single TCP handoff, each handoff still needs extra custom packet to perform the TCP state migration.

Aron et al. (1999) also proposed another solution named back-end request forwarding, back-end forwarding for short. The front-end server is a layer-4 switch for distributing requests from clients. As shown in Fig. 3b, when the connection-handling server does not have the requested data, it would get the data from other servers that have the requested data, and then forward the data back to the client. This mechanism is easy to implement using Network File System, but it would increase the latency time for request processing and consume internal network bandwidth.

Sit et al. (2004) proposed a different mechanism named Multiple Cloning, which could make multiple socket cloning between back-ends. As Fig. 3c illustrates, when a back-end could not handle a request, it could migrate the connection to the back-end that has the requested data using the Socket Cloning technique.

2.4. Existent content-aware request distribution policies

The content information embedded in the HTTP request packet is helpful for developing more sophisticated dispatching policies. The main idea of Client-aware Dispatching Policy (CAP) (Casalicchio and Colajanni, 2001) is to share all classes of services among the back-ends. It classifies requests from clients into four classes, namely normal, CPU-bound, disk-bound, and disk- and CPU-bound. When a HTTP request packet arrives, the front-end parses the requested URL and distinguishes which class it belongs to, and then schedules the packet in this class with the Round-Robin algorithm.

The goals of Locality-aware Request Distribution (LARD) (Pai et al., 1998) are to increase performance by improving the cache hit ratio in back-end servers and achieving load balance. It logically partitions the workload among the back-ends in order to optimize the usage of the overall cluster RAM. Therefore, the requests for the same web page will be distributed to the same back-end node that most likely has the file cached in the main memory, unless the back-end is overloaded.

Workload-aware Request Distribution Strategy (WARD) (Cherkasova and Karlsson, 2001) assigns a small set of most frequently accessed files, namely core files, to be served locally by any back-end nodes in the web cluster system, and partitions the rest of the files, namely partition files, to be served by different back-end nodes. The goal of WARD strategy is to minimize the forwarding overhead due to TCP handoff for the most frequently accessed files. For example, if one back-end receives a request for a core file or local partition file, it would process the request itself. On the contrary, if the request is for a remote partition file, it would handoff this request to the designated back-end, and that back-end would process the request and respond to the client.

2.5. Other related works

Yang and Luo (1999) proposed a cluster architecture that uses a two-way layer-7 front-end for dispatching requests in Linux. To transfer TCP connections efficiently, the front-end pre-forks a number of TCP connections to each server and these pre-forked connections will be bounded to user connections for connection transfer. Based on this architecture, URL Formalization was proposed to effectively support request distribution and the reliability issue is discussed in their later work (Yang and Luo, 2001). HACC proposed by Zhang et al. (1999a,b) is also a two-way layer-7 cluster architecture designed for locality enhancement and dynamic load balancing. In a two-way cluster architecture, the inbound and outbound packets of back-ends should be modified and passed through the front-end, which would consume system resources of the web cluster. Therefore, in the recent work of Luo et al. (2005), they also work on the one-way layer-7 web switch.

Andreolini et al. (2003) proposed a cluster architecture that uses the one-way layer-7 web switch based on the TCP Handoff approach, called ClubWeb-1w. They implemented a new communication protocol, i.e. THOP, in the standard TCP/IP stack in Linux for TCP connection transfer. Additional packet transmission for connection transfer is still needed. Their performance results demonstrate that with careful design and implementation, a content-aware web switch can be extremely scalable.

To support content-aware request distribution in LVS, Wang (2004) has started the TCPHA (TCP HAndoff) project which is a sub-project of LVS. It also implements a kernel-level one-way layer-7 front-end for Linux. It is based on TCP handoff, and TCP connection state transfer is still needed between front-end and back-end during TCP handoff processing. The packet routing technique used is IP tunneling, which is less efficient than direct routing. Besides, service partition is used such that each back-end is responsible for dedicated service such as HTM server, GIF/JPG server, and DVD server.

For scalability of content-aware request distribution in web clusters, Aron et al. (2000a,b) proposed a cluster architecture, in which the TCP connection establishment and handoff are distributed over all back-ends, rather than being centralized in the front-end. Their architecture uses a layer-4 front-end for dispatching incoming requests to back-ends, in which the request dispatching policies do not consider the requested content. The chosen back-end may forward the incoming request to another back-end by handing off the connection using the TCP handoff protocol to another back-end according to the requested content.

Sit et al. (2004) proposed Socket Cloning, which is a distinct connection transfer mechanism in which the connection transfer component is designed in the back-end nodes rather than in the front-end. Their architecture uses a layer-4 front-end for dispatching incoming requests to back-ends. With Socket Cloning technique, the back-end nodes could move an opened socket between the back-ends. For each Socket Cloning, the native socket node (i.e. connection-handling back-end node) has to transfer an extra socket cloning message for cloning a socket and forward the ACK packet to the cloned socket node (request-handling back-end node). These extra packet transfers would consume network bandwidth. However, the back-ends can send response packets directly to clients.

Park et al. (2001) proposed the inter-backend prefetch algorithms in web servers, in which each back-end node predicts the next HTTP requests and prefetches the requested web objects from local disks or other back-end nodes into server RAM. In this way, most requested web objects are directly accessed from the back-end node's main memory rather than from its local disk or other back-end nodes in the cluster. The performance of the web cluster could thus be improved.

Content placement and resource management are also important issues in designing web clusters. Our work can be enhanced by several studies to address these issues. For example, Aron et al. (2000a,b) proposed the Cluster Reserves mechanism for global resource management in cluster-based server systems. Shen et al. (2002) designed an integrated resource management framework for cluster-based services with service differentiation support. Luo et al. (2002) presented a content management system for web clusters with layer-7 routing and content segregation on multiple nodes. Zhuge and Li (Zhuge, 2007; Zhuge and Li, 2007) focus on the research of decentralized content management.

3. LVS-CAD/FC cluster with content-aware dispatching and file caching

LVS-CAD/FC is a web cluster that can perform content-aware request distribution with file caching mechanism. It uses a kernel-level one-way layer-7 web switch to dispatch requests from clients and prefetches a small set of most frequently accessed files into server 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. Web contents are prefetched into servers' file-based caches in RAM because web requests tend to ask for a whole file, whereas the buffer cache of a traditional file system caches the individual blocks of a file rather than the whole file in the RAM. With the file prefetching method, we could make sure that the whole file would be cached in the RAM. Prefetching web contents could avoid data transferring between RAM and disk for those prefetched files, thus reducing disk access overhead. Besides, the size of the file-based cache which affects how many files can be prefetched can be configured by administrator.

The LVS-CAD/FC web cluster system employs various techniques as follows. The TCP Rebuilding technique, a light-weight connection transfer mechanism, is applied in our content-aware dispatcher. The fast Multiple TCP Rebuilding, an efficient and extended version of TCP Rebuilding technique, is implemented to efficiently support HTTP/1.1 persistent connection in the web cluster. The CWARD/CR and CWARD/FR strategies described later in Section 4 that achieve better memory usage and a high cache hit ratio of the web cluster while balancing load among servers are proposed and implemented. A light-weight method to redirect requests from clients to cached files and a mechanism of prefetching web contents into server RAM are implemented to reduce disk I/O times.

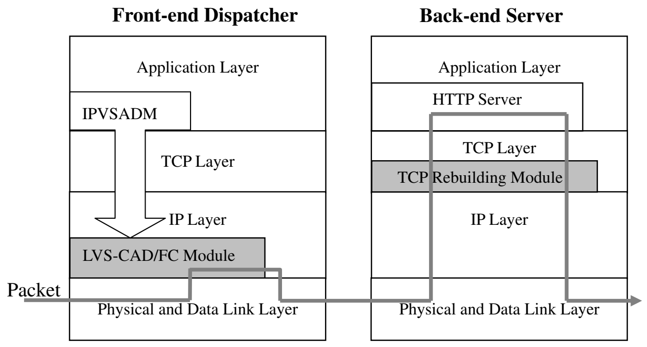


Fig. 4. Kernel modules of LVS-CAD/FC.

This section describes the implementation of our LVS-CAD/FC cluster. Section 3.1 introduces the overview, architecture, and operations of LVS-CAD/FC cluster. The light-weight request redirection method for redirecting requests from clients to cached files is described in Section 3.2. The fast Multiple TCP Rebuilding to provide efficient support for persistent connection is implemented and described in Section 3.3.

3.1. Architecture and operations of LVS-CAD/FC web cluster

The LVS-CAD/FC web cluster, as shown in Fig. 4, requires the operating system in the front-end to be configured with a loadable LVS-CAD/FC kernel module and the TCP codes in the back-end nodes to be customized with the TCP Rebuilding module.

The operations of LVS-CAD/FC are depicted in Fig. 5 and described below.

1. The client first initiates a TCP connection with the front-end and then sends the HTTP request to the front-end.
2. After the forward module of the front-end node receives the request packet, it uses the client IP, client port, and the protocol of this packet as the hash key, and then looks up the connection table to verify if the connection has been established.

3a.&4. If the connection is not established, the forward module would then call the dispatcher module to schedule the request packet using the designated request scheduling algorithm. The request scheduling algorithm is responsible for choosing one back-end from the corresponding set of back-ends that may serve this request according to the proposed content-based workload-aware request distribution policies. In particular, the dispatcher module could use all request scheduling algorithms provided in LVS. The dispatcher module then updates the connection table with this new connection, and then calls the LVS-CAD/FC module.

3b. If the connection record of the request packet exists, the forward module would start the LVS-CAD/FC module directly, bypassing the dispatcher module.

5. The LVS-CAD/FC module looks up the URL table according to the URL in the request packet, and then verifies if there are back-ends having cached the requested file.
6. If no servers have cached the requested data, the request packet would be forwarded to the chosen back-end without modification. Otherwise, the URL of the request would be modified to the corresponding path name where the requested data or file is stored. Then, the LVS-CAD/FC module would check if multiple handoff using the Multiple TCP

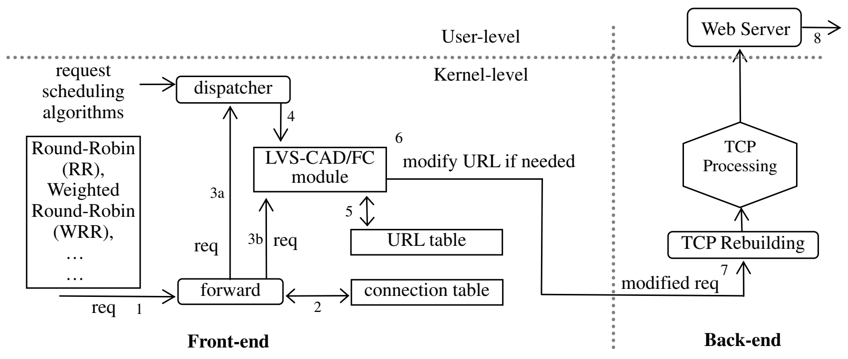


Fig. 5. Operations of LVS-CAD/FC.

Rebuilding technique is needed, such that the request packet can be forwarded to the proper back-end that has the requested data in RAM.

7. When the selected back-end node receives the request packet, it would rebuild the connection using the TCP Rebuilding technique.
8. After the web server receives the request packet, it processes and generates the requested data, and then responds directly to the client, bypassing the front-end.

3.2. Request redirection

We have implemented a simple and efficient request redirection technique to efficiently redirect the HTTP request to the file-based cache in the main memory of back-end directly by modifying the URL, as shown in Fig. 6.

In each back-end, we use *ramfs* to preserve a certain amount of memory as the file-based cache to store the most frequently accessed files inside. The URL table in the front-end records the corresponding information where files are located. When receiving a request, the front-end would look up the URL table and check if there are back-ends having cached the requested file. If true, it would change the URL of this request for redirecting it to the file-based cache in the memory of the proper back-end. The performance of the caching system is thus improved.

3.3. The fast Multiple TCP Rebuilding

The TCP Rebuilding technique (Liu et al., 2007) 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 to the request of the client directly, bypassing the front-end node. To provide efficient support for HTTP/1.1 persistent connection, the TCP Rebuilding technique is extended to Multiple TCP Rebuilding, as shown in Fig. 7, 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.

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 perform 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 have implemented the fast 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 Fig. 8b, rather than generating and sending it before handing off the request to the different back-end as shown in Fig. 8a. The data flow of Fig. 8b is described as follows:

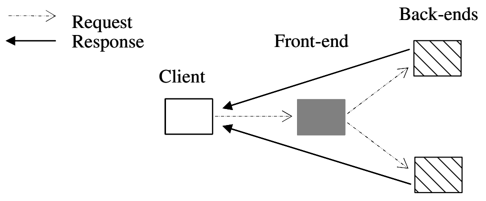


Fig. 7. Multiple TCP Rebuilding.

1–3:	The front-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 using the designated request scheduling algorithm according to the proposed content-based workload-aware request distribution policies 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, process the request, and finally forward 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 back-end 2 using Multiple TCP Rebuilding technique. After the connection is rebuilt in back-end 2, a RST packet is generated and sent to back-end 1 to disconnect the obsolete connection.
12–14:	After back-end 2 rebuilds the connection with the client, it would respond to the client directly and the subsequent packets would be forwarded to back-end 2.

The Multiple TCP Rebuilding incurs the cost including the overheads for the front-end to generate a RST packet to disconnect with the precious back-end server and modify the proper connection record, for back-end 1 to tear down the connection when it receives the generated RST packet, and for back-end 2 to rebuild the connection with the client using the TCP Rebuilding technique. Because of these overheads, Multiple TCP Rebuilding should be

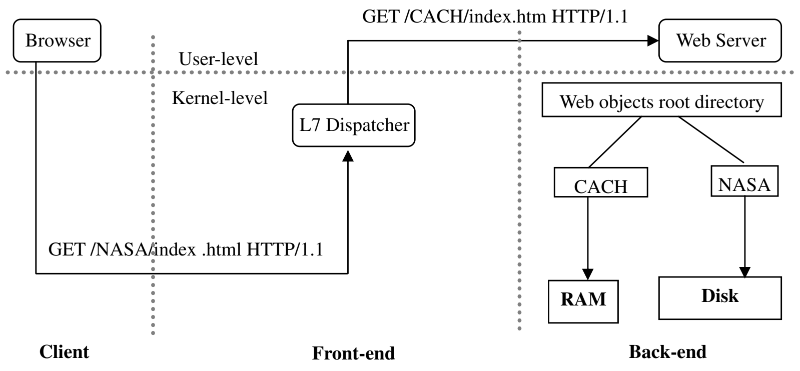
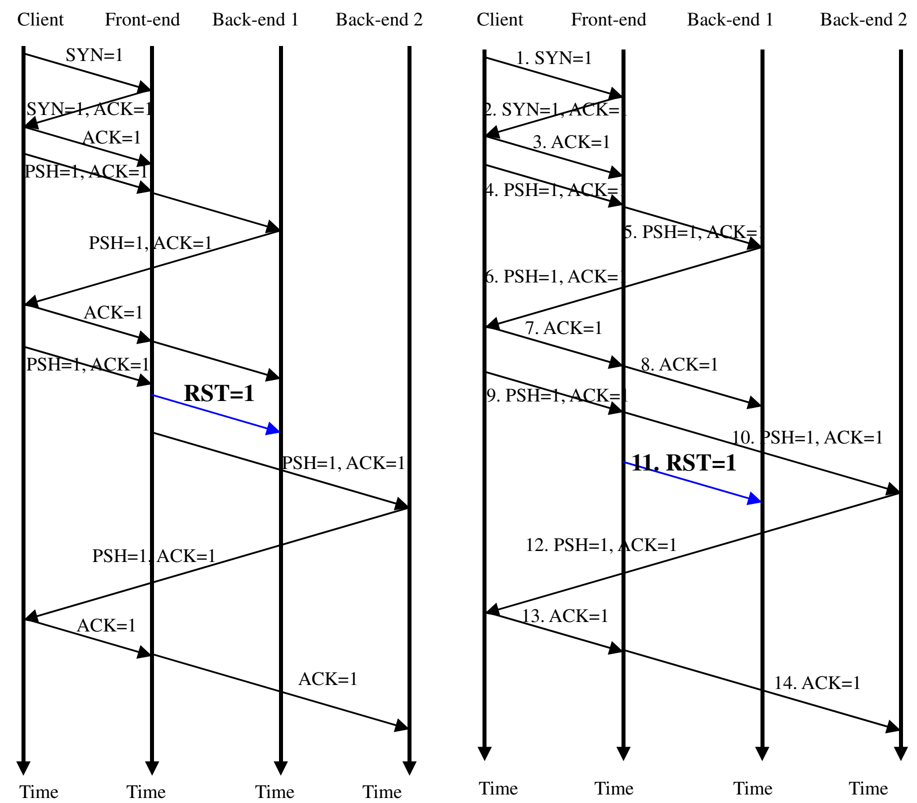


Fig. 6. Request redirection.



(a) Sending RST packet before connection is migrated

(b) Sending RST packet after connection is migrated

Fig. 8. Packet forwarding flow of multiple TCP Rebuilding.

used only when it is beneficial, in this case, for improving the cache hit ratio.

4. Proposed content-based workload-aware request distribution policies

4.1. Motivation

Several studies (Andreolini et al., 2003; Aron et al., 1999, 2000a,b; Cardellini et al., 2002; Casalicchio and Colajanni, 2001; Cherkasova and Karlsson, 2001; Lin et al., 2003; Liu et al., 2007; Pai et al., 1998; Park et al., 2001; Sit et al., 2004; Wang, 2004; Yang and Luo, 1999; Zhang et al., 1999a,b) have shown that content-aware request distribution that takes content in requests into account when dispatching requests can achieve more effective resource utilization of web cluster. In fact, being aware of workload information also helps request dispatching (Arlitt and Williamson, 1997; Cherkasova and Karlsson, 2001; Markatos, 1996).

In Arlitt and Williamson's study (1997) of web server workload, they identified 10 common characteristics in their collected data sets. One important characteristic is the high concentration of references in the web server. In their study, caching 10% of the most frequently accessed files consumed only 6–45 MB main memory size, and 80–96% cache hit ratio could be achieved. Markatos also obtained similar findings in his study (Markatos, 1996). 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's cache hit ratio at the cost of only a small amount of main memory size.

WARD strategy (Cherkasova and Karlsson, 2001) also takes advantage of workload characterization that assigns the most frequently accessed files to be served by each back-end node locally to minimize the forwarding overhead incurred from TCP handoff for those most frequently accessed files. Since the workload characterization of web traffic influences significantly the performance of web service, workload characterization should thus be taken into consideration when designing a web cluster.

Since in a web cluster all requests from clients are sent to the front-end, if the front-end is content-aware and is able to examine the content of HTTP request, it can easily collect the reference information and identify the most frequently accessed web files, and then decide the set of back-ends that are suitable for serving the request.

Due to these considerations, we propose and present new Content-based and Workload-aware Request Distribution policies, named CWARD/CR and CWARD/FR. Both strategies take content of request and workload characteristic into account in dispatching requests. For these two policies, the accessed web files with high reference count are prefetched into servers' file-based cache in memory. Basically, the files with more access frequencies would be duplicated in more servers' RAM, whereas, the duplication methods as described in Sections 4.2 and 4.3 are different for these two policies. The common goals are to achieve better memory utilization and better cache hit rates in servers while achieving better load balancing among servers. Therefore, in LVS-CAD/FC cluster, the front-end examines the content of request (i.e. URI) and analyzes workload characterization according to the proposed policies in dispatching requests from clients to servers.

Since access frequencies of web files are changing all the time, the file-based caches in servers' RAM should be updated to react to the changes. In LVS-CAD/FC, the file-based caches are updated according to the proposed CWARD/CR or CWARD/FR policies when the system is idle or on a daily basis. The overheads incurred from prefetching the distinct web files are expected to be small because previous research (Arlitt and Williamson, 1997) shows that the distinct bytes of the requested data in the period of a day are small in most traces.

4.2. Content-based workload-aware request distribution with core replication – CWARD/CR

The proposed CWARD/CR (i.e. CWARD with Core Replication) policy is shown in Fig. 9. Similar to the WARD strategy, we identify a small set of most frequently accessed files, named core, and a set of less frequently accessed files to be partitioned among back-ends, 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. If the request belongs to the cached files (i.e. core or part files), the URI in the content of request packet is modified to correspond with the path where the target cached file is stored. Then the dispatcher forwards the request to the corresponding back-end node.

The CWARD/CR policy works as follows. First of all, the core files and part files are prefetched into RAM of back-ends, and the URL table in the front-end is then 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, Target A in Fig. 9, the front-end will choose a back-end according to the designated request scheduling algorithm and modify the URI in the content of the 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 the 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, the front-end would then handoff the request to the back-end that has the requested data. Lastly, if the requested web file belongs neither to the core set nor the part set, the front-end forwards the request to the chosen back-end according to the assigned request scheduling algorithm without packet modification.

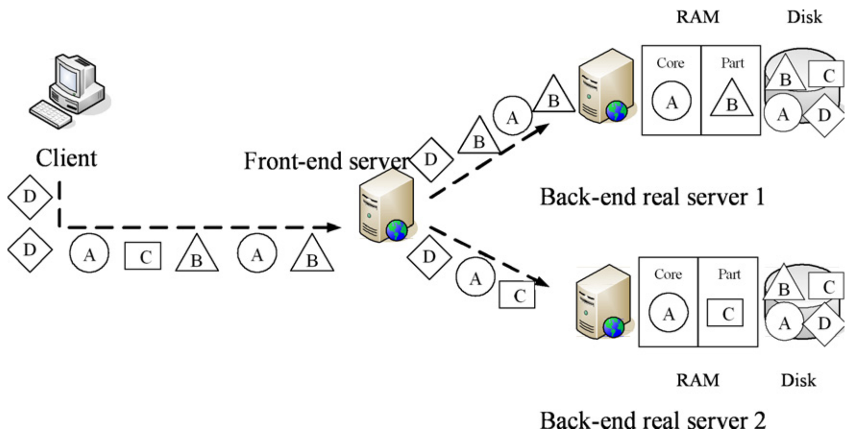


Fig. 9. Dispatching policy of CWARD/CR.

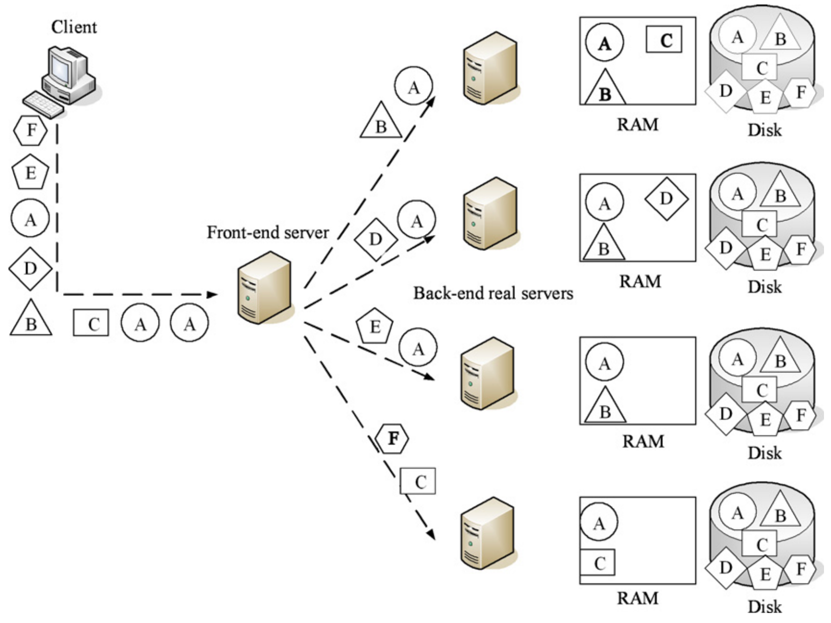


Fig. 10. Dispatching policy of CWARD/FR.

4.3. *Content-based workload-aware request distribution with frequency-based replication – CWARD/FR*

The proposed CWARD/FR (i.e. CWARD with Frequency-based Replication) policy is shown in Fig. 10. Similar to the CWARD/CR strategy as described in Section 4.2, a small set of frequently accessed files are prefetched into back-end nodes. The difference is that the most frequently accessed files are replicated in the most of back-ends, whereas, the lesser frequently accessed files are replicated in the lesser amount of back-ends. The basic idea is to let files be cached in back-ends according to their access frequencies, such that the more frequently files are accessed, the more back-ends can be selected to serve these frequently accessed files.

Since CWARD/FR takes workload properties into account, CWARD/FR policy first determines the number of back-ends that can be selected to serve a specific web file using the following formula:

$$\frac{AccessFrequencyi}{\max\{AccessFrequencyi|i=1..n\}} * TotalNumOfServers \tag{1}$$

The $AccessFrequency_i$ denotes the number of times the specific web file i was accessed during a period. In our experiments presented later in Section 5.4, the value of *AccessFrequency* is normalized by the $\log_{10}$ function. We use the maximum frequency among all accessed files as the denominator instead of using the summation of the frequency times of accessed files as the denominator. This is because we want the web files with the highest access frequencies can be cached in each back-ends' file-based cache, such that the hottest web files can be served by the most of back-ends. After the number of back-ends that can be selected to serve a specific web file is determined, then the front-end uses the Round-Robin manner to assign a set of back-ends for caching this web file. The content-aware dispatcher examines the content information (i.e. URI) in each request. If the request belongs to the cached files, the URI in the content of request packet is modified to correspond with the path where the target cached file is stored. Then the dispatcher forwards the request to the selected back-end node according to the designated request scheduling algorithm.

For example, targets A–D in Fig. 10 are frequently accessed files, so they are prefetched into the server RAM according to formula 1. Among them, Target A is the hottest one, so it is prefetched into each back-end's file cache. Because of the difference of access frequencies, the amount of replication of these frequently accessed files is different. So the goals of CWARD/FR are to achieve better load balancing among back-ends and better memory utilization than CWARD/CR does while increasing cache hit rates of back-ends.

When the front-end receives a sequence of requests, it will examine the content of the HTTP request. If the requested content belongs to the web file prefetched, e.g. the target A in Fig. 10, the front-end will modify the URI in the content of the HTTP request to appropriate path for redirecting the HTTP request to the designated back-end's file cache according to the designated request scheduling algorithm. Whereas, if the requested content is not prefetched, e.g., the target E or F in Fig. 10, the front-end routes the HTTP request to the chosen back-end according to the designated request scheduling algorithm without packet modification.

5. Performance evaluation

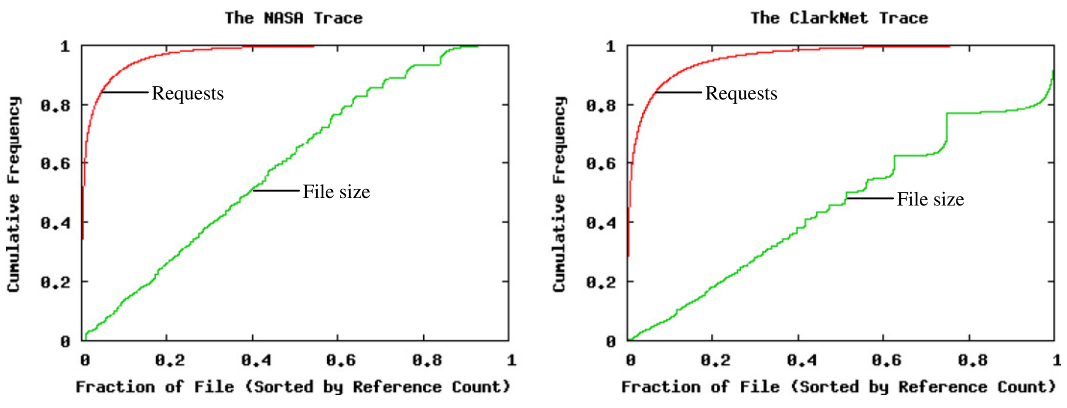
In this section, we present performance evaluation of our proposed web cluster system. Section 5.1 provides the software and hardware configuration of our experimental environment. Section 5.2 describes the two real-world traces used in our experiments. Section 5.3 presents the evaluation of overheads caused by our system. The results show that our content-aware request distribution incurs only little overhead to the web cluster. Finally, Section 5.4 presents the experimental results with trace-driven benchmarking.

5.1. *Experimental environment*

Our test bed consists of one front-end node, eight back-end nodes, and 10 clients, connected to a single 24-port fast-Ethernet switch. The environment is a stand-alone local area network with no disturbance from external network traffic. The packet forwarding mechanism is set to be direct routing (Mack, 2003) and the request scheduling algorithm is set to be Weighted Round-Robin (WRR) (Linux, 2006; Zhang et al., 1999a,b). Table 1 lists the hardware/software components.

Table 1
Hardware/software environment

	Front-end	Back-end	Client
Processor (MHz)	Intel P4 3.4G		Intel P4 2.4G
Memory (MB)	DDR 256	DDR 256/128	DDR 256
NIC (Mbps)	Intel Pro 100/1000		Reltek RTL8139
OS	Red Hat Linux 8.0		Windows XP/Red Hat Linux 8.0
Kernel	2.4.18		SP1/2.4.18–14
IPVS	1.0.4	X	X
Web server	X	Apache 2.0.40	X
Benchmark	X		WebBench 5.0/http_load



(a) NASA trace (b) ClarkNet trace

Fig. 11. Locality of references.

The benchmarks used for measuring the performance of web servers are WebBench (2005) and Http_load (2006). WebBench 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 employed 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 randomly. Moreover, because there are 10 clients in our test bed, we split the log into 10 parts in the Round-Robin manner, with each part for a single client.

5.2. Access logs

We use two publicly available traces from the Internet Traffic Archive (Internet, 2000), namely traces of NASA Kennedy Space Center and ClarkNet web servers. The working sets used in this research are derived from these two logs. The NASA trace contains 3,092,291 successful requests in the two-month period and a total of 62,483 MB bytes were transferred (Arlitt and Williamson, 1997). This trace needs 40 MB main memory size to achieve 96% cache hit ratio. Moreover, the size of the data set derived is 245 MB. The ClarkNet trace spanned two weeks, during which there were 2,940,873 successful requests and a total of 27,592 MB bytes were transferred. It requires 34 MB main memory size to achieve 90% cache hit ratio. The size of the data set derived is 467 MB.

Fig. 11 shows the cumulative distribution of request frequency and size for NASA and ClarkNet traces. The files were sorted in decreasing order of request frequency.

5.3. Overhead evaluation of the front-end

In this experiment, we set up the WebBench to repeatedly request the same web page of a given size in order to measure the

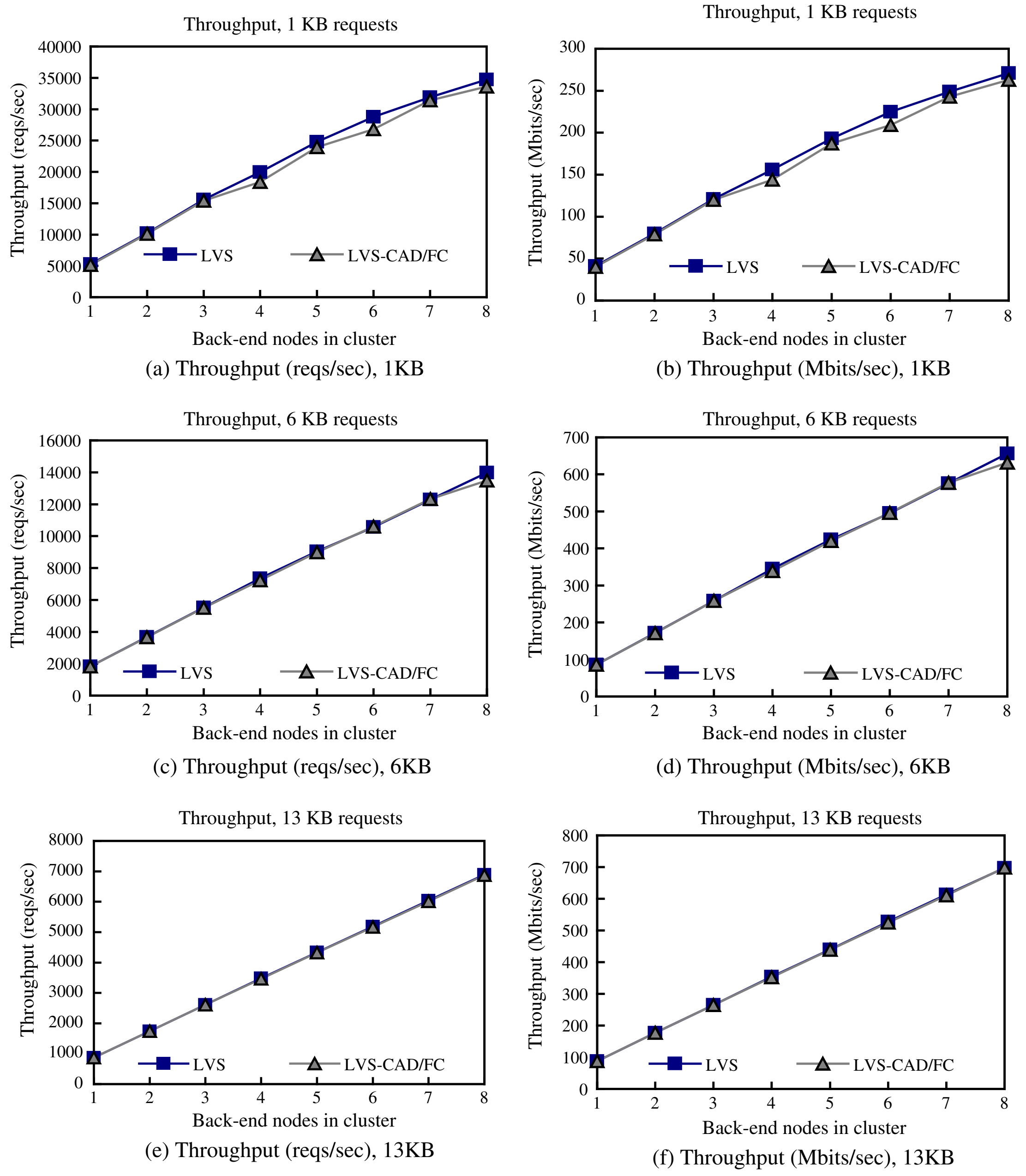


Fig. 12. Experimental results of overhead evaluation.

overheads incurred by our proposed system. The requested web pages are of 1 KB, 6 KB and 13 KB, respectively. The 1 KB web page is chosen because the entire HTTP response could be transferred in a single Ethernet frame. The 6 KB and 13 KB web pages are chosen because their sizes correspond to the average HTTP transfer sizes reported in the literature (Arlitt and Williamson, 1997). The number of back-ends ranges from one to eight.

We do not set up the LVS-CAD/FC to prefetch the target web page into RAM in this experiment, since our purpose is to investigate the additional overheads caused by this content-aware dispatching web system as compared with a content-blind web system. The additional overheads would include examining the URL, looking up the URL table, and modifying the URL. The content-blind web cluster system, LVS, is used for comparison.

As shown in Fig. 12, the LVS-CAD/FC cluster scales as well as the LVS cluster. The overheads incurred by our content-aware web cluster, LVS-CAD/FC, only degrade performance slightly. LVS-CAD/FC performs only 3.2% less than the LVS in the 1 KB experiment, 3.5% less in the 6 KB experiment, and 0.3% less in the 13 KB experiment.

Fig. 13 compares the CPU idle time between the content-aware front-end of the LVS-CAD/FC and the content-blind front-end of

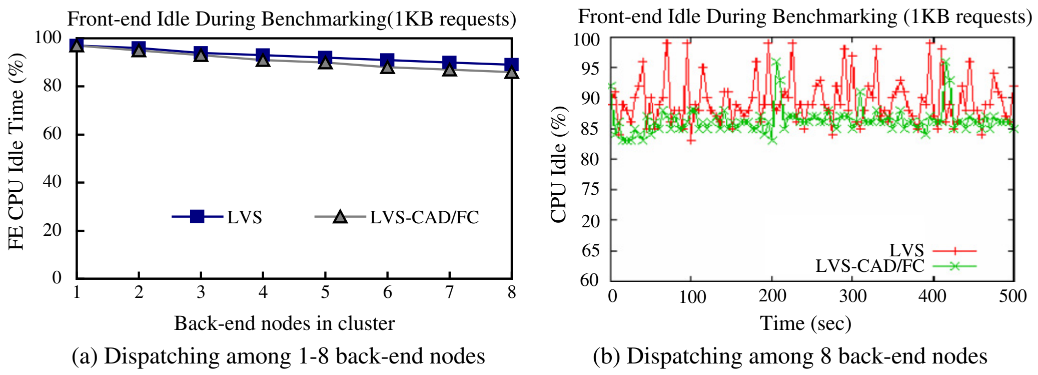
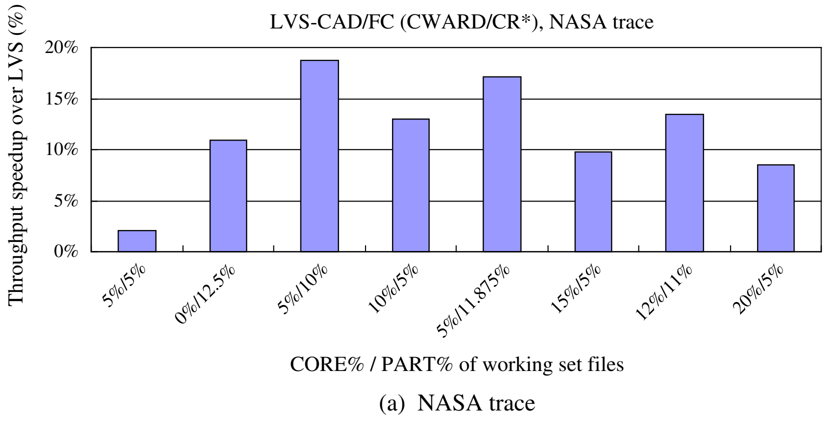


Fig. 13. Front-end idle time during benchmarking.



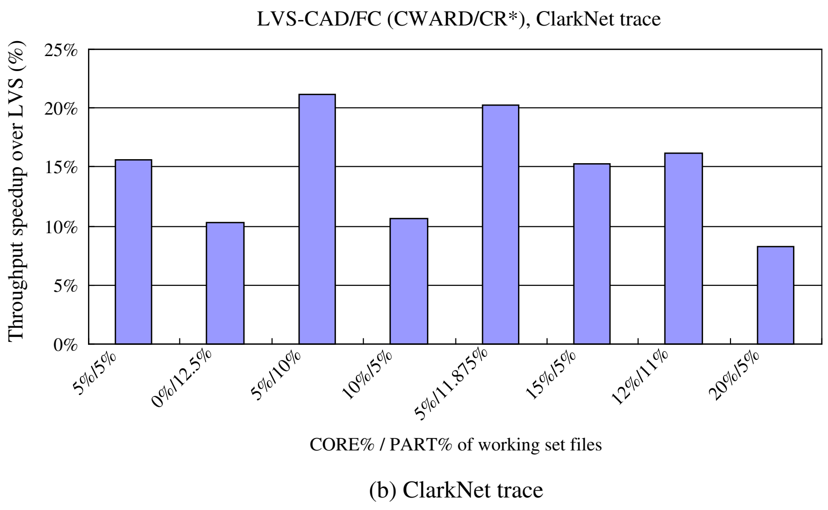


Fig. 14. Trace-driven benchmarking with CWARD/CR*.

LVS. The result shows that the front-end nodes of these two systems are far from being a system bottleneck, and the content-aware front-end of our LVS-CAD/FC consumes only a little more CPU computation than the content-blind front-end of LVS.

From the above experiments, we can conclude that LVS-CAD/FC platform is efficient and its performance is comparable with that of the content-blind LVS. However, LVS-CAD/FC is further capable of performing content-aware request dispatching. Therefore, more intelligent policies for improving cluster performance can be applied.

5.4. Trace-driven benchmarking

We evaluate our LVS-CAD/FC cluster with the proposed content-based workload-aware request distribution policies. The benchmark used is http_load and two realistic workloads derived from the logs of NASA and ClarkNet are used. Eight back-end nodes are used and each back-end node is configured with 128 MB RAM in the experiments with NASA trace while configured with 256 MB RAM in the experiments with ClarkNet trace.

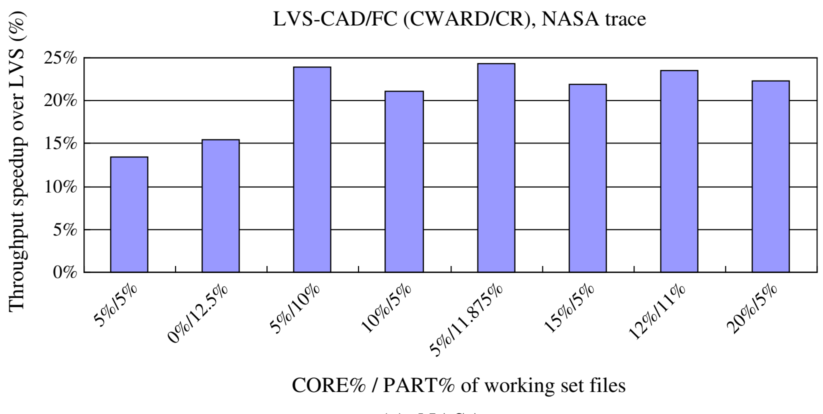
Sections 5.4.1 and 5.4.2 presents performance results obtained with our LVS-CAD/FC cluster and the proposed CWARD/FR and CWARD/CR policies. Section 5.4.3 presents the scalability test and the results demonstrate that our LVS-CAD/FC cluster scales well and outperforms the LVS web cluster with a layer-4 dispatcher.

5.4.1. Experimental results of CWARD/CR

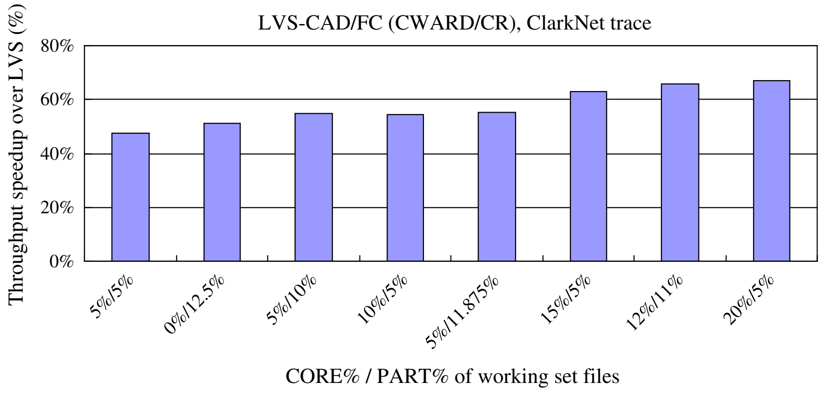
Figs. 14 and 15 show the results of trace-driven benchmarking with CWARD/CR policy and the throughput speedup over the baseline LVS is measured. The *x*%/*y*% (core%/part%) means that the core set (i.e. the most frequently accessed files) has *x*% of the working set files, and the part set (i.e. the less frequently accessed files) in each node also has *y*% of the working set files. In addition, LVS-CAD/FC (CWARD/CR*) means that the LVS-CAD/FC platform adopts the WARD-like strategy, and LVS-CAD/FC (CWARD/CR) means that the LVS-CAD/FC platform adopts the CWARD/CR policy. The difference between LVS-CAD/FC (CWARD/CR) and LVS-CAD/FC (CWARD/CR*) is that each back-end using CWARD/CR has prefetched about $(x + y)$% of the working set files into the file-based cache in the RAM, but the back-end using CWARD/CR* has not. In other words, though back-ends using CWARD/CR* do not prefetch web files into RAM, the front-end still dispatches requests belonging to core set to one of the back-ends according to the designated request scheduling algorithm, and dispatches requests belonging to part set to the pre-assigned back-end.

Fig. 14 shows that LVS-CAD/FC (CWARD/CR*) outperforms LVS by 2.03–18.7% and 8.28–21.19% respectively in the NASA trace and ClarkNet trace among different combinations of core and part sets. These results demonstrate that even without prefetching, with effective request distribution policy, web cluster can make use of limited RAM more effectively. Because disk I/Os is reduced, performance is thus improved.

Fig. 15 shows that with prefetching, performance can be further improved since disk I/Os are further reduced. LVS-CAD/FC (CWARD/CR) outperforms LVS by 13.41–24.39% and 47.68–66.89% in the NASA trace and ClarkNet trace, respectively. These results also show that when only about 15% of working set files (i.e. about 36.8 MB in NASA trace and 70 MB in ClarkNet trace) are prefetched into each back-end's cache, performance can be improved by 24% in NASA trace and 55% in ClarkNet trace.



(a) NASA trace



(b) ClarkNet trace

Fig. 15. Trace-driven benchmarking with CWARD/CR.

5.4.2. Performance comparison of various request distribution policies

Fig. 16 shows the results of comparing various request distribution policies. In this experiment, CWARD/CR and CWARD/CR* use 12%/11% combination of the core and part sets. That is, back-ends using CWARD/CR have prefetched about 23% of the working set files (i.e. about 56.4 MB in NASA trace and 107.4 MB in ClarkNet trace) into RAM. In addition, the difference between CWARD/FR and CWARD/FR* is only that back-ends using CWARD/FR* do not prefetch web files into RAM. The results show that LVS-CAD/FC (CWARD/FR) has the best performance. In Fig. 16a, though back-ends of LVS-CAD/FC using CWARD/FR and CWARD/CR strategies prefetch similar size of web files (i.e. about 56.4 MB) into file cache, LVS-CAD/FC (CWARD/FR) outperforms LVS and LVS-CAD/FC (CWARD/CR) by 36.18% and 10.2% respectively in the NASA trace. Even without prefetching, LVS-CAD/FC (CWARD/FR*) still outperforms LVS and LVS-CAD/FC (CWARD/CR*) by 19.92% and 5.73%, respectively.

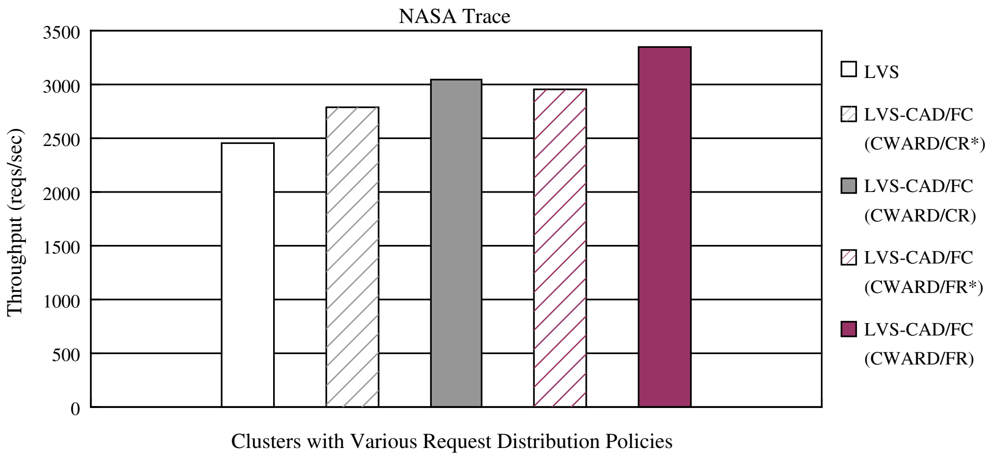
In Fig. 16b, LVS-CAD/FC (CWARD/FR) outperforms LVS by 65.89% in the ClarkNet trace. In this experiment, though LVS-CAD/FC (CWARD/FR) and LVS-CAD/FC (CWARD/CR) have similar performance, however, back-ends of LVS-CAD/FC using CWARD/FR strategy prefetch about 37% less of web files (i.e. about 67.7 MB) into file cache than using CWARD/CR strategy. Even without prefetching, LVS-CAD/FC (CWARD/FR*) still outperforms LVS by 16.23%.

In summary, these results demonstrate that with the effective request distribution policies such as WARD and CWARD (i.e. CWARD/CR* and CWARD/FR*), the web cluster with content-aware request distribution can effectively outperform the web cluster with content-blind request distribution. Prefetching files into cache as done in CWARD/CR and CWARD/FR, which effectively reduces disk I/Os, can further greatly improve web cluster performance.

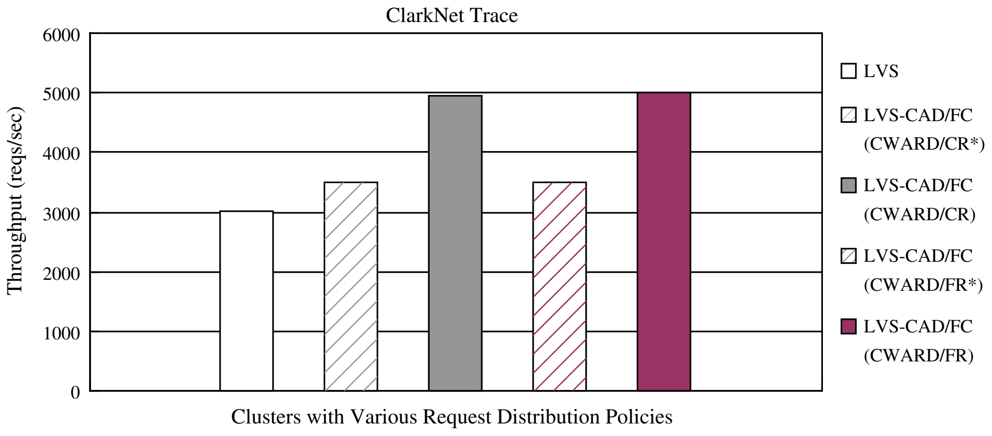
Besides, these results also demonstrate that CWARD/FR performs much better than CWARD/CR and prefetches significantly less amount of web files into the file-based cache. This is because in CWARD/FR, the most frequently accessed files cached in back-ends are set according to their access frequencies instead of being cached in each back-end, such that CWARD/FR can make use of RAM more effectively than CWARD/CR or WARD-like policy does while achieving better load balance and better cache hit rates of back-ends.

5.4.3. Scalability analysis

In this experiment, we evaluate the scalability of our LVS-CAD/FC system with the proposed CWARD/FR content-aware request distribution policy. Each back-end using CWARD/FR has prefetched about 23% of the working set files (i.e. about 56.4 MB) in NASA trace and about 14.5% of the working set files (i.e. about 67.7 MB) in ClarkNet trace into RAM. The benchmark used is http_load and the number of back-ends ranges from one to eight. Fig. 17 shows that LVS-CAD/FC scales well and outperforms LVS by 36.18% in the NASA trace and 65.89% in the ClarkNet trace when the number of back-ends is eight.



(a) NASA trace



(b) ClarkNet trace

Fig. 16. Performance of various request distribution policies.

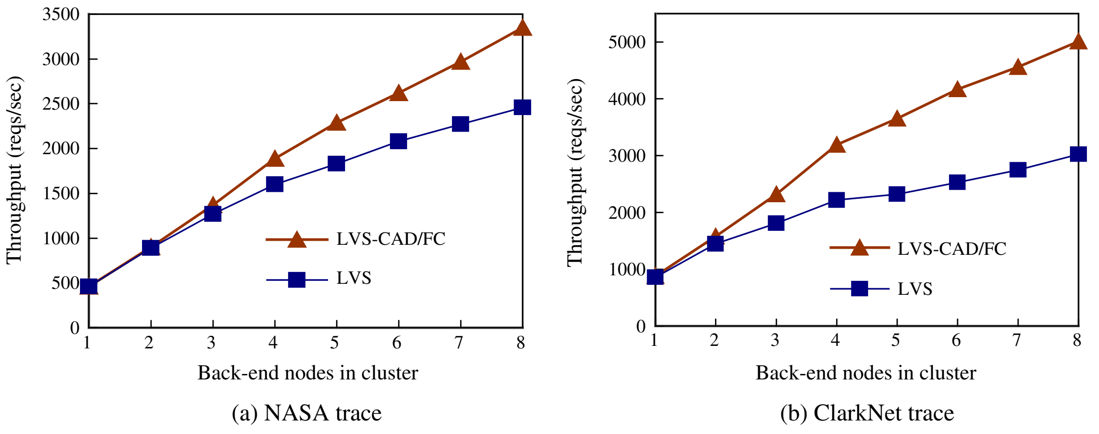


Fig. 17. Scalability test.

6. Conclusion

Basing on our prior work (Lin et al., 2005), we have implemented a high-performance and scalable content-based web cluster named LVS-CAD/FC. The LVS-CAD/FC cluster implements a file-based web cache to cache a small set of the most frequently accessed web files in RAM and implements a kernel-level content-based front-end to distribute HTTP requests from clients among back-end nodes. With one-way architecture, the back-ends can send requested data directly back to clients, bypassing the front-end. Besides, the fast Multiple TCP Rebuilding is implemented to efficiently support persistent connection. Besides being a high-performance web cluster, LVS-CAD/FC system can also serve as the research platform for content-aware request distribution policies.

In this paper, we have proposed new policies, which consider both the content of requests and workload characterization in dispatching requests for better memory utilization and increasing the cache hit ratio and load sharing of the web cluster. For CWARD/CR, the most frequently accessed files are cached in each back-end and the less frequently accessed files are partitioned among back-ends with the Round-Robin manner. For CWARD/FR, files cached in back-ends are set according to their access frequencies, so that the more frequently files are accessed, the more back-ends can be selected to serve these frequently accessed files.

Experimental results of practical implementation on Linux show that the LVS-CAD/FC cluster with the kernel-level one-way content-based web switch is efficient and scales well. Moreover, the trace-driven benchmarking with the working sets derived from the logs of NASA and ClarkNet demonstrates that our LVS-CAD/FC cluster with the proposed content-aware request distribution policies can achieve 36.18% and 66.89% better performance than the layer-4 LVS web cluster respectively. Experimental results also show that CWARD/FR substantially outperforms CWARD/CR since CWARD/FR can utilize RAM more effectively and can achieve better load balancing among back-ends than CWARD/CR does.

Further research is needed for optimizing the way of prefetching web pages in the server RAM. In addition, how to efficiently prefetch dynamic web contents into server RAM is another issue that needs to be investigated. Based on this platform, several issues could also be further explored, such as cooperative caching, support of quality of service, and adaptive content-aware dispatching algorithms.

References

Andreolini, M., Colajanni, M., Nuccio, M., 2003. Kernel-based web switches providing content-aware routing. In: Second IEEE International Symposium on Network Computing and Applications, Cambridge, MA, April 16–18.

Arlitt, M.F., Williamson, C.L., 1997. Internet web servers: workload characterization and performance implications. IEEE/ACM Transactions on Networking 5 (5), 631–645.

Aron, M., Druschel, P., Zwaenepoel, W., 1999. Efficient support for P-HTTP in cluster-based web servers. In: Annual USENIX Technical Conference, Monterey, CA, June.

Aron, M., Sanders, D., Druschel, P., Zwaenepoel, W., 2000a. Scalable content-aware request distribution in cluster-based network servers. In: Annual USENIX Technical Conference, San Diego, CA, USA, June 18–23.

Aron, M., Druschel, P., Zwaenepoel, W., 2000b. Cluster reserves: a mechanism for resource management in cluster-based network servers. In: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA, June, pp. 90–101.

Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.S., 2002. The state of the art in locally distributed web-server systems. ACM Computing Surveys 34 (2), 263–311.

Casalicchio, E., Colajanni, M., 2001. A client-aware dispatching algorithm for web clusters providing multiple services. In: Proceedings of the 10th International World Wide Web Conference, Hong Kong, May 1–5, pp. 535–544.

Cherkasova, L., Karlsson, M., 2001. 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, June, pp. 212–221.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., 1999. RFC2616, Hypertext Transfer Protocol – HTTP/1.1, June. <http://www.w3.org/Protocols/>.

Http_load, 2006. <http://www.acme.com/software/http_load/>.

Internet Traffic Archive Website, 2000. <http://ita.ee.lbl.gov/>.

Lin, Y.D., Tsai, P.T., Lin, P.C., Tien, C.M., 2003. Direct web switch routing with state migration, TCP masquerade, and cookie name rewriting. In: Global Telecommunications Conference, San Francisco, December, vol. 7, pp. 3663–3667.

Lin, Y.C., Chiang, M.L., Guo, L.F., 2005. System support for workload-aware content-based request distribution in web clusters TANET 2005, TaiTung, Taiwan, ROC. This paper also appears in. Journal of Internet Technology 7 (3), 261–267.

Linux Virtual Server Website, 2006. <http://www.linuxvirtualserver.org/>.

Liu, H.H., Chiang, M.L., Wu, M.C., 2007. TCP Rebuilding for content-aware request dispatching in web clusters. Software Practice & Experience 37 (11), 1215–1241.

Luo, M.Y., Yang, C.S., Tseng, C.W., 2002. Content management on server farm with layer-7 routing. In: The 17th ACM Symposium on Applied Computing, Spain, March.

Luo, M.Y., Yang, C.S., Tseng, C.W., 2005. Analysis and improvement of content-aware routing mechanisms. IEICE Transactions on Communications E88-B (1), 227–238.

Mack, J., 2003. LVS-HOWTO, July. <http://www.linuxvirtualserver.org/Joseph.Mack/HOWTO/index.html>.

Maliz, D., Bhagwat, P., 1998. TCP Splicing for Application Layer Proxy Performance. IBM Technical Report RC-21139, March.

Markatos, E.P., 1996. Main memory caching of web documents. In: Proceedings of the Fifth International World Wide Web Conference, Paris, France.

Pai, V.S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E., 1998. Locality-aware request distribution in cluster-based network servers. In: Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October.

Park, S.Y., Park, D., Lee, J., Cho, J.W., 2001. Efficient Inter-backend Prefetch Algorithms in Cluster-based Web Servers. HPC Asia, September.

Shen, K., Tang, H., Yang, T., Chu, L., 2002. Integrated Resource Management for Cluster-based Internet Services. In: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA, pp. 225–238.

Sit, Y.F., Wang, C.L., Lau, F., 2004. Cyclone: a high-performance cluster-based web server with socket cloning. Cluster Computing 7 (1), 21–37.

Steven, C., et al., 1999. Method and System for Directing a Flow between a Client and a Server. US Patent 6,006,264, December 21.

Wang, L., 2004. TCPHA Project Website, April 28. <http://dragon.linux-vs.org/~dragonfly/>.

WebBench Website, 2005. <http://www.etestinglabs.com/benchmarks/Webbench/Webbench.asp>.

Yang, C.S., Luo, M.Y., 1999. Efficient support for content-based routing in web server clusters. In: 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, Colorado, USA, October 11–14.

Yang, C.S., Luo, M.Y., 2001. System support for scalable, reliable, and highly manageable web hosting service. In: 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, California, USA, March 26-28.

Zhang, W., Jin, S., Wu, Q., 1999a. Creating Linux Virtual Servers. In: LinuxExpo 1999 Conference.

Zhang, X., Barrientos, M., Chen, J.B., Seltzer, M., 1999b. HACC: an architecture for cluster-based web servers. In: Proceedings of the 3rd USENIX Windows NT Symposium Seattle, Washington, USA, July 12–13.

Zhuge, H., 2007. Autonomous semantic link networking model for the Knowledge Grid. Concurrency and Computation: Practice and Experience 19 (7), 1065–1085.

Zhuge, H., Li, X., 2007. Peer-to-peer in metric space and semantic space. IEEE Transactions on Knowledge and Data Engineering 19 (6), 759–771.

Mei-Ling Chiang received the B.S. degree in Management Information Science from National Chengchi University, Taipei, Taiwan, in 1989. She received the M.S. degree in 1993 and her Ph.D degree in 1999 in Computer and Information Science from National Chiao Tung University, Hsinchu, Taiwan. Now she is an Associate Professor in the Department of Information Management at National Chi-Nan University, Puli, Taiwan. Her current research interests include operating systems, embedded systems, and clustered system.

Yu-Chen Lin received his M.S. degree in Information Management from National Chi Nan University, Puli, Taiwan, in 2005. He now works in the Chunghwa Telecom Co., Ltd., Taiwan. His research interests include networks and operating systems.

Lian-Feng Guo received his M.S. degree in Information Management from National Chi-Nan University, Puli, Taiwan, in 2006. His research interests include distributed systems, Internet technologies, and clustered systems.