An Efficient Web Cluster with Content-Aware Request Distribution for Streaming Service

Cheng-Wei Lin
Department of Information Management
National Chi Nan University
Email: s96213514@ncnu.edu.tw

Mei-Ling Chiang
Department of Information Management
National Chi Nan University
Email: joanna@ncnu.edu.tw

Chen-Yu Yang
Department of Information Management
National Chi Nan University
Email: s97213503@ncnu.edu.tw

***Abstract*—**As the advance of the web and multimedia technologies, the web sites usually can provide not only static and dynamic web page services but also multimedia streaming services. To serve various types and huge amount of service demands, nowadays the web cluster system has been popularly deployed because of the advantages of cost effectiveness, load sharing, scalability, and high availability.

Traditionally, for providing the multimedia streaming service, the server needs to establish a streaming connection with the client at first and then transmits the streaming data. The servers need to maintain the connection until the entire streaming service is finished. In the process of providing streaming service, the servers cannot handoff the existing connection to the other server to continue the streaming service. This might cause load unbalance among servers and degrade the performance of the whole web cluster. In this paper, we have proposed and implemented the RTSP Handoff mechanism. Using our proposed mechanism, a long runtime film can be logically partitioned into several sections and each section of the film can be served by different real server. During the process, the client would not sense the change of servers. In this way, a web cluster can achieve better load balance when providing the streaming service.

We have implemented our mechanism in the Linux kernel of the LVS-CAD web cluster. Experimental results demonstrate that the LVS-CAD web cluster with our proposed RTSP Handoff mechanism can reduce 34.35% average response time and achieve 37.19% better throughput than the one without our proposed mechanism when providing multiple web services.

***Index Terms*:** Web Cluster, Multimedia Streaming, Content-aware Request Distribution

1. Introduction

With the fast development of internet and the advance of the web and multimedia technologies, the web services become more diverse. The web services are not limited to static pages, dynamic pages, and streaming services. To provide multiple web services needs more hardware resources, so the traditional single server is not sufficient to handle such heavy workload. The web cluster that is composed of a front-end request dispatching server and several back-end request-handling servers has become a cost-effective way to serve huge amount of service demands, because of the advantages of load sharing and load balance, high performance, scalability, and high availability.

For providing the multimedia streaming service, the streaming server needs to establish a streaming connection with the client and then starts to transmit the streaming data. In general, no matter how long the movie runtime is, the server has to maintain the connection with the client until the entire streaming service is finished. For example, if a movie runtime is an hour, the streaming server needs to provide the streaming service and maintain the connection for an hour. However, the streaming server might be overloaded with heavy streaming workload from lots of clients. In the process of providing streaming service, the servers cannot handoff the existing connection to the other server to continue the streaming service. This might cause load unbalance among servers and degrade the performance of the whole web cluster.

In this paper, we have proposed a new mechanism named RTSP Handoff. Using our proposed mechanism, a long runtime film can be logically partitioned into several sections and each section of the film can be served by different real server. During the process, the client would not sense the change of servers. In this way, a web cluster can achieve better load balance when providing the streaming service.

We have implemented the proposed RTSP Handoff mechanism in the LVS-CAD web cluster [1,2]. The LVS-CAD web cluster is a Linux-based cluster-based web server system which originally provides only static and dynamic web pages. By applying the proposed mechanism, LVS-CAD web cluster can efficiently support streaming service. Experimental results demonstrate that the LVS-CAD web cluster with our proposed RTSP Handoff mechanism can achieve 10.99-37.19% better throughput under the heavy streaming workload, and 12.85-29.55% better throughput under the synthetic workload than the one without using our proposed mechanism. Besides, the average response time of the LVS-CAD web cluster when using our proposed RTSP Handoff mechanism can be reduced by 15.54-34.35% under the heavy streaming workload, and reduced by 6.94-34.59% under the synthetic workload than the one without using our proposed mechanism.

2. Background and Related Work

This section introduces the system background and related work.

2.1 Streaming Service Protocol

2.1.1 Real Time Streaming Protocol (RTSP)

The Real-Time Streaming Protocol (RTSP) is a one of network control protocol for real-time streaming data based on TCP protocol. It was developed by the Multiparty Multimedia Session Control Working Group (MMUSIC WG) of the Internet Engineering Task Force (IETF) and published in RFC 2326 in 1999. The RTSP protocol is used to configure the sessions between client and server. It does not encode any content of message. It's only responsible for settings and control message, similar to HTTP. All RTSP requests are request-response pair to complete every communication. The functionalities of RTSP protocol are described in the following and Figure 1 shows the packet flow of the streaming service.

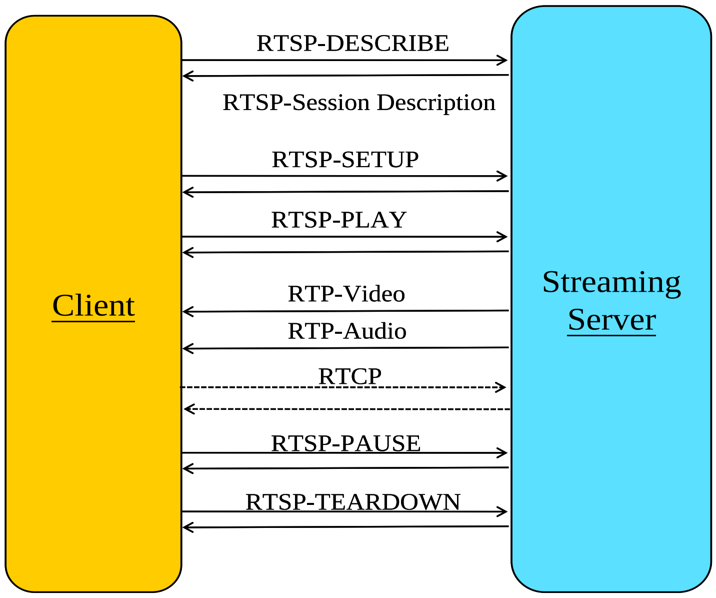

Figure 1. Packet flow of the streaming service

(1) **DESCRIBE**: The RTSP-DESCRIBE request includes film name, format, language, and player version from the request made by client. Then the streaming server finds the corresponding information including film length, track types and IDs. The player on client will set film according to this information.

(2) **SETUP**: Because each video and audio data have their own track number included in RTP packet transmitted by the streaming server. The RTSP-SETUP request is used to inform the streaming server of the receive port number with corresponding track number. The streaming server would return its port number, path of film source and a session number of the connection.

(3) **PLAY**: After the previous two steps, the client could make the RTSP-PLAY request time to start playing the film. It includes the initial and terminal time. When receiving the RTSP-PLAY request, the streaming server will return the timestamp that fills in the film track ID, and the initial sequence number of the RTP packet. After that, the streaming server transmits the streaming data on the RTP protocol.

(4) **PAUSE** is used to suspend a playing movie.

(5) **TEARDOWN** request can terminate a streaming connection.

2.1.2 Real Time Transport Protocol (RTP)

This protocol is defined by the Audio-Video Transport Working Group of the IETF and first published in 1996 as RFC 1889 and superseded by RFC 3550 in 2003. It carries streaming media based on the UDP protocol to the client by the way of unicast, multicast or broadcast. Using the RTP protocol can obtain better efficiency and avoid the delay of movie playing. The RTP protocol does not provide the control mechanism for transmitting the streaming data. For this reason, the RTP Control Protocol (RTCP) could provide the control mechanism for the RTP packet flow. The client can send the RTCP packets periodically to inform the streaming server to change transfer rate dynamically. Figure 2 illustrates the RTP header format.

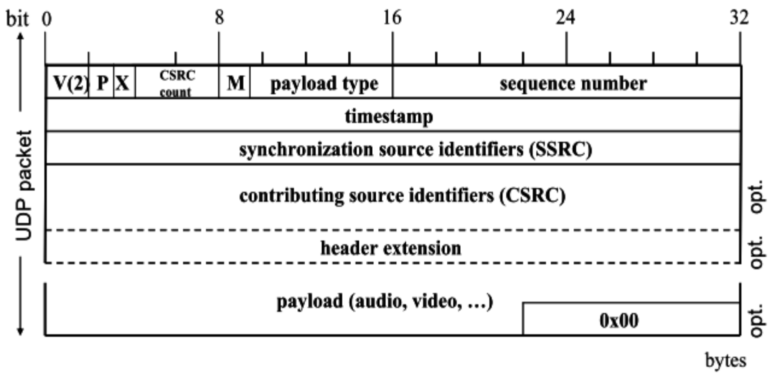

Figure 2. The RTP header format

Some important fields are listed as follow:

(1) Payload type field

This field indicates the data format of RTP packet and determine the decoding method of RTP packet. The payload type of the video streaming data is 96 and the payload type of the audio streaming data is 97. When receiving the RTP packet, the multimedia player will distinguish the payload type corresponding to its value to decode it.

(2) Sequence number field

This field presents the order of RTP packets sent by streaming server. The initial value of the sequence number is generated in random and incremented by one when each RTP packet is sent. The clients could use the sequence number to detect lost packets and reorder the order of RTP packets.

(3) Timestamp field

This field is used to make the client to playback the received RTP packet at appropriate time intervals. The streaming server gives the timestamp corresponding to the time value in the film for each RTP packet. Because the RTP is a real-time transfer protocol, all received RTP packets must be processed immediately. If the timestamp value of the received RTP packet is delayed, the RTP packet would be dropped by client.

2.1.3 Real Time Control Protocol (RTCP)

The RTP Control Protocol (RTCP) is used to control the RTP packet flow. Its basic functionality and packet structure are defined in RFC 3550 superseding its original standardization in 1996 (RFC 1889). When the streaming connection is established, the client will open two ports, one is used to receive the RTP packets, and the other is used to receive the RTCP packets. The client will periodically send RTCP packets to inform DSS the status of receiving video data, which includes the amount of packets received and the amount of lost packets. DSS can use the information to change transfer rate dynamically.

2.1.4 MPEG-4 Part 14 (MP4) File Format

MPEG-4 Part 14 is a multimedia container format standard specified as a part of MPEG-4 by ISO/IEC, which is based on Apple's QuickTime container format. A MP4 file is basically comprised of video and audio streams.

Note that before the streaming server start to transmit the media data, it must be encapsulated to streaming packets with a hint track. A hint track stores corresponding information into the packet with appropriate size and then sends to the client. Figure 3 shows the MP4 file structure.

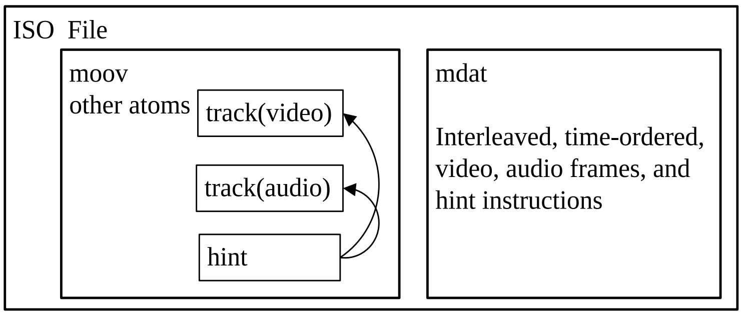
ISO File
moov
other atoms
track(video)
track(audio)
hint
mdat
Interleaved, time-ordered, video, audio frames, and hint instructions

Figure 3. The MP4 file structure diagram with a hint track

2.2 DSS System Overview

DSS stands for Darwin Streaming Server also called QuickTime Streaming Server (QTSS) [6], which is released by Apple Computer Inc. in 1999. This is the first open-source software of streaming server. DSS can install on many operating systems such as Linux, Solaris and Windows. It also supports MPEG-4 video format, so the service of Video on Demand and live service can use it to transmit multimedia data.

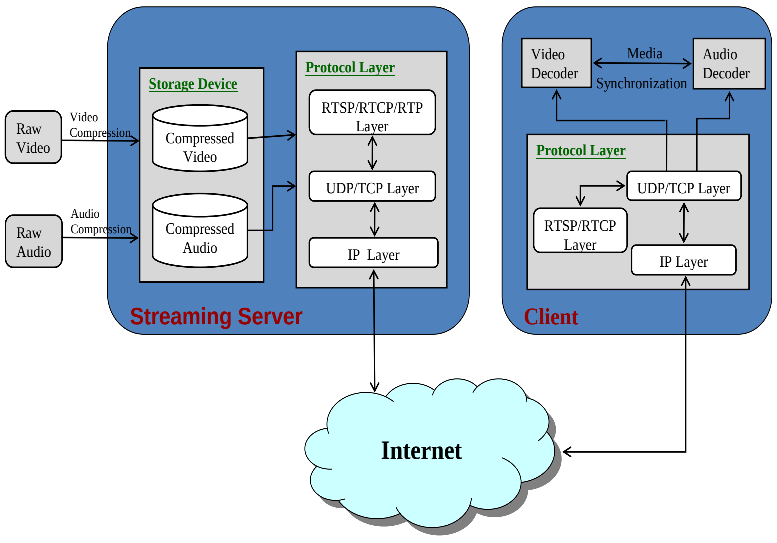

Figure 4. The client-server communication architecture in DSS

Figure 4 illustrated the client-server architecture in DSS. At first, the client makes a RTSP request to initialize a streaming connection to the DSS. After the connection is established, the client could send different RTSP methods to play, pause and stop the playing movie [5]. Subsequently, the DSS returns the streaming data by RTP packets. While the film is playing, the client sends the RTCP packets to the DSS, including the number of packets received or lost periodically. According to this information, the DSS could change the transfer rate in transmission.

2.3 Web Cluster System

Since a single server cannot efficiently handle huge amount of requests, a web cluster system is a good deployment to handle this situation. We will briefly discuss some aspects we have used.

2.3.1 Linux Virtual Server (LVS)

LVS [4] is one of the most popular web cluster system which contains a set of independent Linux-based servers and acts as a single server. It is composed of a front-end server and multiple back-end servers. The front-end server is a layer-4 web switch which can perform only content-blind request distribution that cannot parse the content of the HTTP request (i.e. URL) in dispatching requests from clients to back-end servers.

LVS supports three types of packet forwarding techniques, network address translation, IP tunneling and direct routing. In this paper, we choose direct routing as our environment of experiment, because it's the most efficient mechanism.

In the direct routing mechanism, the front-end server and back-end servers have the same virtual IP address (VIP). The front-end server routes a packet to the selected back-end server directly by modifying MAC address. Therefore, all servers must be linked in continuous LAN segment. The direct routing mechanism belongs to one-way packet rewriting architecture, in which the back-end servers respond all requests to clients directly. The front-end would not become a bottleneck because it processes only incoming packets. In order to prevent both of the front-end server and back-end servers from returning the ARP response packets to clients at the same time, back-end servers should disable the ARP response. Figure 5 shows the packet forwarding flow of LVS with direct routing.

2.3.2 Layer-7 Web Switch Mechanism

The layer-7 web switch which works at application level can support content-aware routing which is more sophisticated than content-blind request distribution to make the back-end servers achieve better load balancing. This routing mechanism is less efficient than the layer-4 web switch because it has to parse the content of the requests. In order to conduct content-aware request distribution, the front-end server needs to do three-way handshaking with the client for receiving the packet containing the HTTP content.

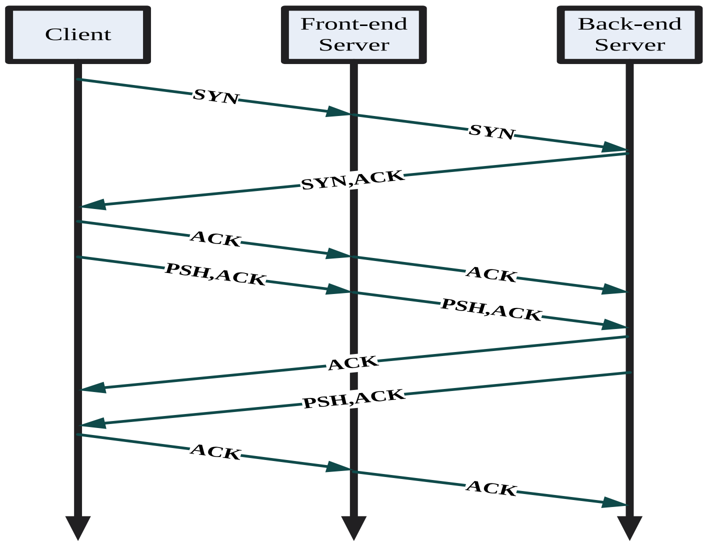

Figure 5. Packet forwarding flow of LVS with direct routing mechanism

2.4 LVS-CAD Web Cluster

2.4.1 Content-Aware Request Distribution Policy

Our front-end server uses the content-aware dispatching policy named Grouped Client-Aware Policy (GCAP) [3] to select a back-end server to provide the streaming service. When the front-end server receives the notification packet which is sent by the original back-end server to handoff the current streaming service, it uses the same dispatching policy to select the next back-end server for continuing the streaming service.

The concept of GCAP is based on CAP [7]. Because different types of requests need different resources, CAP classifies the requests from clients into four types including normal (N), CPU bound (CB), disk bound (DB), and disk and CPU bound (DCB) services. Each type of requests from clients will be dispatched to the proper back-end server by using the Round-Robin (RR) policy, so the back-end servers would handle each type of requests evenly. In order to distinguish the processing capabilities of the back-end servers, the GCAP was proposed in our early work to limit the request types that each server could process. GCAP dispatches different types of requests to back-end servers with Weighted Round-Robin (WRR) scheduling, and each back-end server processes only its own types of requests.

Figure 6 shows an example of GCAP policy. We assume there are three back-end servers in the cluster, and both server A and server B can serve type-1 requests while server C can serve only type-2 requests. When the front-end server receives the type-1 requests, it dispatches the type-1 requests to the server A and server B based on WRR policy. Therefore, the server A and server B would handle type-1 requests evenly. Because only server C can serve type-2 requests, so the seventh, eighth, and ninth requests are handled by back-end server C.

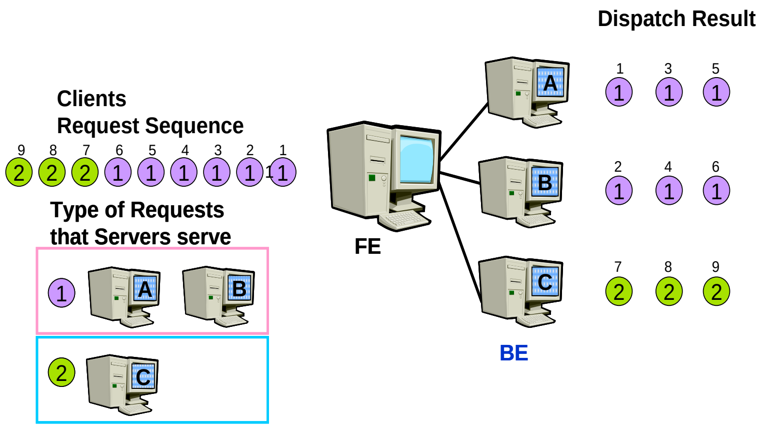

Figure 6. Request sequence and dispatching result of GCAP

2.4.2 The architecture of LVS-CAD

Figure 7 shows the architecture of our LVS-CAD

[1,2] (LVS with Content Aware Dispatching) cluster. To improve content-aware request distribution effectively, we designed a fast handshaking [1] by IPVS-CAD module on the front-end server and the TCP Rebuilding mechanism on each back-end server. The fast handshaking mechanism can do TCP three-way handshaking at IP layer instead of TCP layer to gain better performance than the original three-way handshaking.

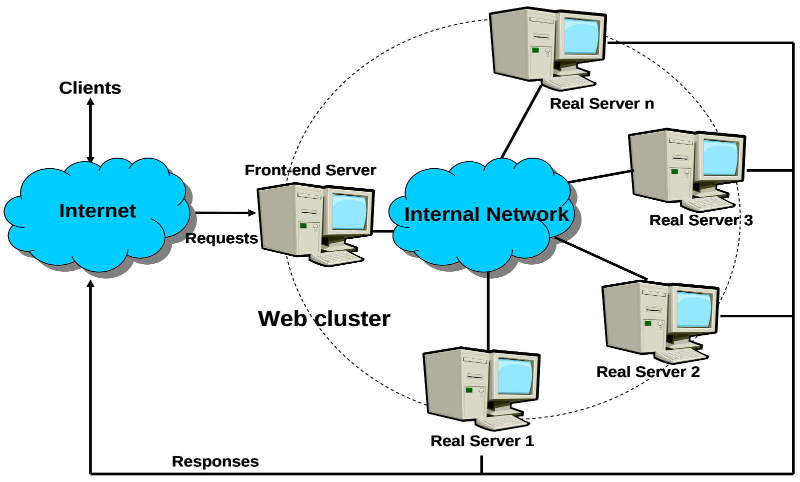

Figure 7. The architecture of LVS-CAD

3. Design and Implementation of a Web Cluster Supporting Streaming Service

This section describe how we design and implement a web cluster to support a variety of web services including static, dynamic web pages and multimedia streaming service. In order to make our front-end server effectively to distribute the loading of multimedia streaming service while the other services are running, we proposed the RTSP Handoff mechanism on our LVS-CAD web cluster system. For this purpose, we additionally propose the RTSP rebuilding method and RTSP handoff request method to support the RTSP Handoff mechanism.

3.1 System Overview

We have constructed the LVS-CAD web cluster as our experiment platform of streaming service. For this purpose, we installed the DSS on all back-end servers.

Now, the client wants to watch a film and make a RTSP request to the front-end server. As shown in Figure 8, the front-end server directs the request to one of back-end servers, and uses the RTSP Handoff mechanism to migrate this on-line connection to the next back-end server that is in lighter workload. Then the next back-end server would rebuild the connection with the client by the RTSP Rebuilding manner.

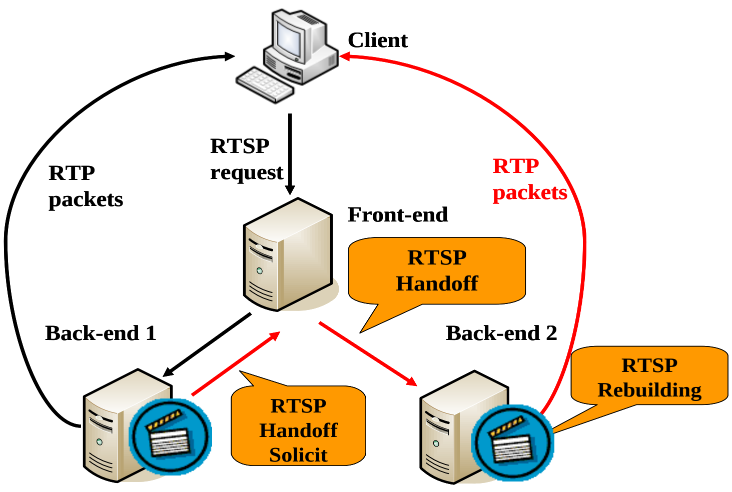

Figure 8. The architecture of our proposed RTSP Handoff mechanism

3.1.1 Processing Flow of RTSP Request

The processing flow of RTSP request is illustrated in Figure 9. The front-end server established a connection efficiently with the client by means of fast handshaking. Then the front-end server forwards the RTSP request to the back-end server according to its designated scheduling algorithm. When receiving the RTSP request, the back-end server uses the TCP Rebuilding mechanism [1] to rebuild the existing TCP connection by changing PSH flag. Finally, the back-end server returns the RTSP reply packet to the client directly (Figure 9). If a subsequent request which belong to the same streaming connection comes in, it would be forwarded to the same back-end server.

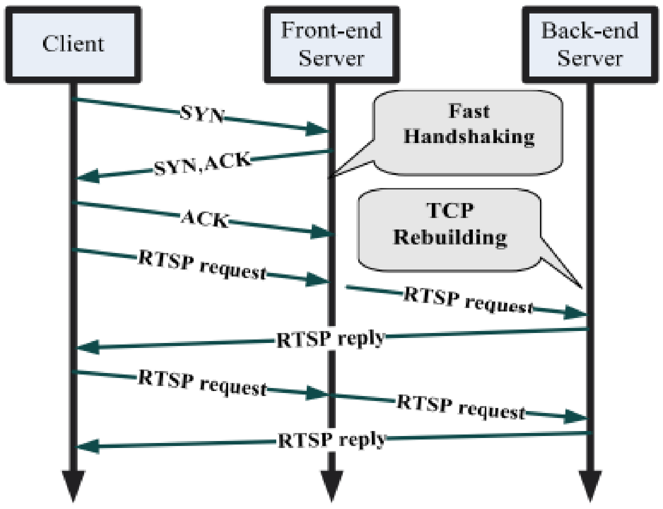

Figure 9. Processing Flow of RTSP Request

3.1.2 Processing Flow of RTP Request

After the streaming connection is set, the client would send the RTSP PLAY request to inform the back-end server to start providing the streaming service, and the back-end server replies the RTSP PLAY response packet to the client. At last, the back-end server transmits the video and audio data in RTP packets separately (Figure 10).

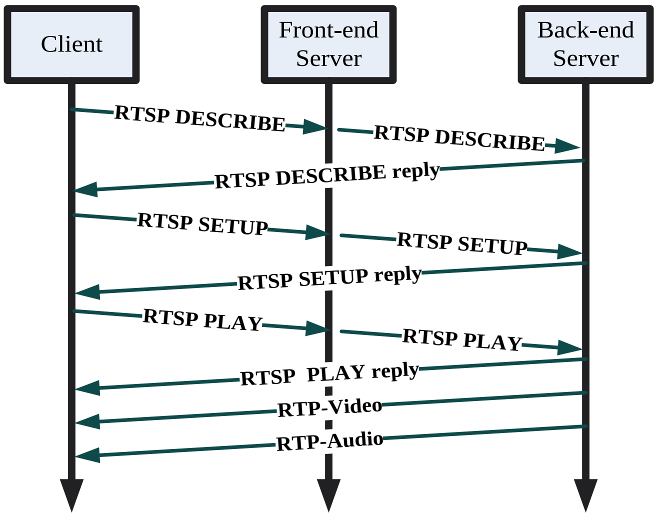

Figure 10. Processing Flow of RTP Request

3.2 Front-end Server Implementation

Due to the streaming connection cannot be disconnected before entire film finished, this may cause load imbalance when the back-end server is in heavy workload. For this reason, we design a method of the front-end server is to logically partition a film into several parts and handoff among back-end servers for lightening their workload.

In principal, establishing a streaming connection only depends on the RTSP DESCRIBE and RTSP PLAY packets. The front-end server uses the client source IP address and port number to insert entries into the RTSP hash table, and determines whether the streaming connection needs to be migrated by the length of the selected film, i.e. whether the film needs to be partitioned.

Figure 11 demonstrate the operation of migrating the existing streaming connection. Before the handoff occurs, the old back-end server sent a notification packet to inform the front-end server to select a new back-end server to continue the streaming service. The notification packet includes the current client source IP address and source port number. Then, the front-end will copy the two packets and send them to the new back-end server for rebuilding the existing connection. According the content of RTSP PLAY packet, the new back-end server can transmit the film with the modified time range of the playing movie to the client. The client would not feel any interrupt during the whole process. The workflow of RTSP Handoff mechanism shows in Figure 12.

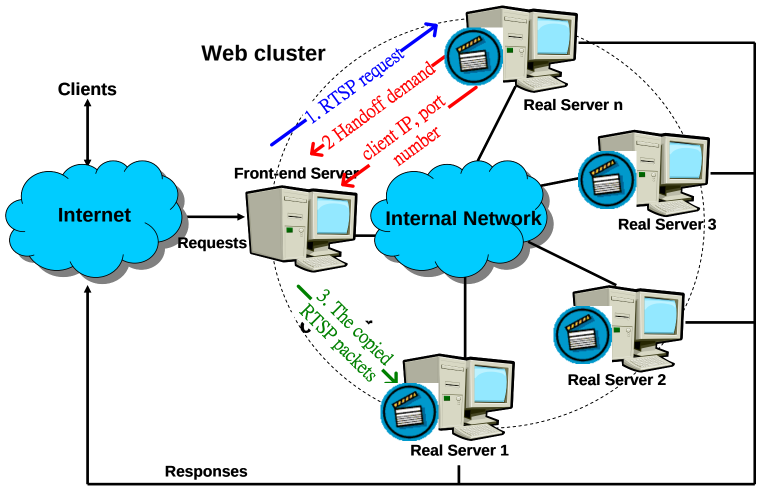

Figure 11. The operation of migrating the existing streaming connection

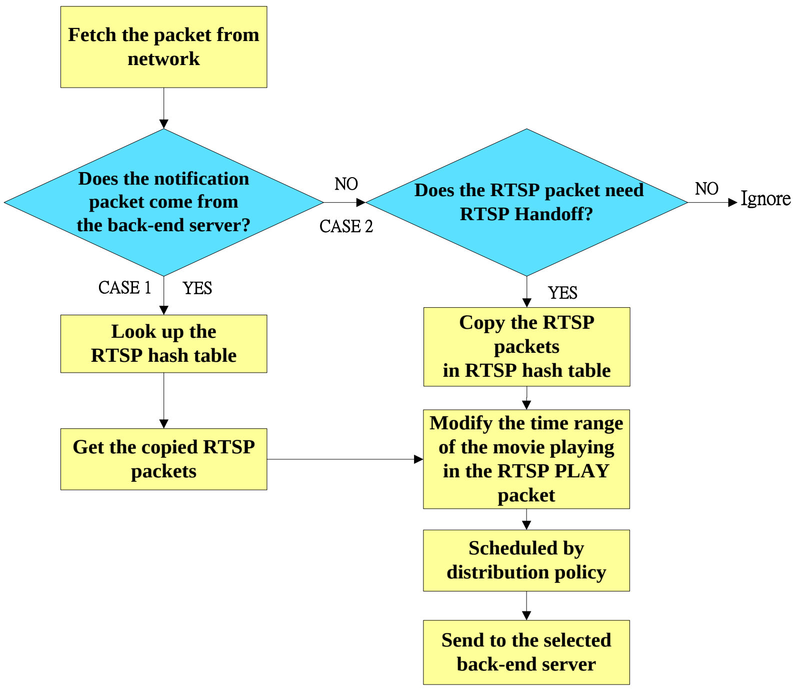
Fetch the packet from network
Does the notification packet come from the back-end server?
NO
CASE 2
Does the RTSP packet need RTSP Handoff?
NO
Ignore
CASE 1
YES
YES
Look up the RTSP hash table
Copy the RTSP packets in RTSP hash table
Get the copied RTSP packets
Modify the time range of the movie playing in the RTSP PLAY packet
Scheduled by distribution policy
Send to the selected back-end server

Figure 12. The flow of RTSP Handoff

3.3 Back-End Server Implementation

When the RTSP SETUP packet comes in, the DSS will generate a session key for identifying streaming connection which the received RTSP request belongs to. Note that the DSS is located at the application layer. While RTSP handoff occurs, the new back-end server receives the copied RTSP SETUP packet, it will generate a new session key for rebuilding connection. However, the subsequent copied RTSP PLAY packets are still going with the old session key of the original back-end server. In order to make this copied RTSP PLAY packet be treated as legal, the new back-end server needs to replace the old session key with the new one at the kernel layer, and then passed it to the application layer. At the same time, the client would not detect that the front-end server has migrated the streaming connection because the back-end server has already dropped the reply packets of the copied RTSP packets. So the client will not receive the extra RTSP reply packets.

When fetching the RTSP SETUP reply packet at the kernel layer, the back-end server uses the client's IP address and the RTSP port number to create a RTSP hash table entry, where the RTSP port number is used to receive the RTSP packets by clients. Then the back-end server stores the RTP port number and the new session key into the entry. The RTP port number is used to receive the RTP packet by the client. RTSP PLAY further uses the RTP port number to create the RTP hash table for the RTP Handoff request method.

When the new back-end server receives the copied RTSP PLAY packet with the old session key, it could use the client source IP address and the RTSP port number to look up new session key from the RTSP hash table, and then replace the old session key with the new one in the packet. It means that RTSP Rebuilding is successful.

Figure 13 shows the RTSP packet flow in function ip_rcv() in Linux kernel. At first, the incoming packet (step 1) has to be examined whether it is the stored RTSP packet. If it's not, the packet is delivered to the upper layer (step 2.a). Otherwise (step 2.b), if the RTSP packet has the session key (step 3.b), the back-end server should find out the entry in RTSP hash table (step 4) to get the new one. If the RTSP packet has no session key, it is delivered to the upper layer (step 3.a). If the back-end server cannot find the entry in the RTSP hash table, it would drop the RTSP packet (step 4.b). Finally (step 4.a), the back-end server replaces the old session key with new one to the packet (step 5), then the packet is passed to the upper layer.

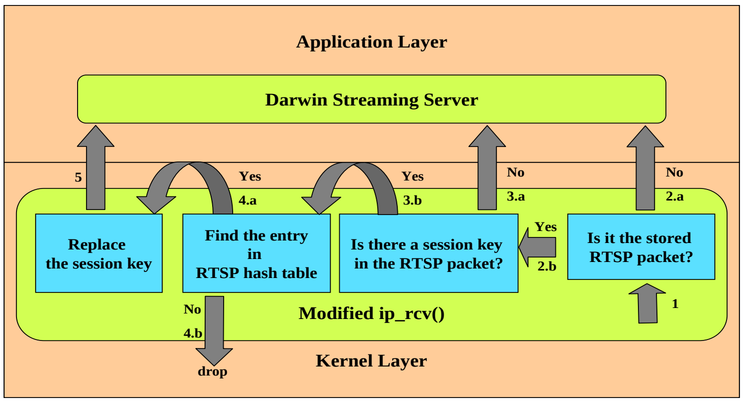

Figure 13. Packet flow in ip_rcv() function in the back-end server

We also added the checkpoint by modifying function ip_finish_output() in Linux kernel for RTSP rebuilding as shown in Figure 14. The incoming packet (step 1) has to be examined to whether it is a duplicate RTSP reply packet. If it's not, the packet is delivered to the client (step 2.a). Otherwise (step 2.b), if it is the RTSP SETUP reply packet (step 3.b), the back-end server should create a RTSP hash table entry then store the new session key and RTP port number. In order to avoid returning the duplicate reply packet to the client, the back-end server drop it at all (step 3.a or 4).

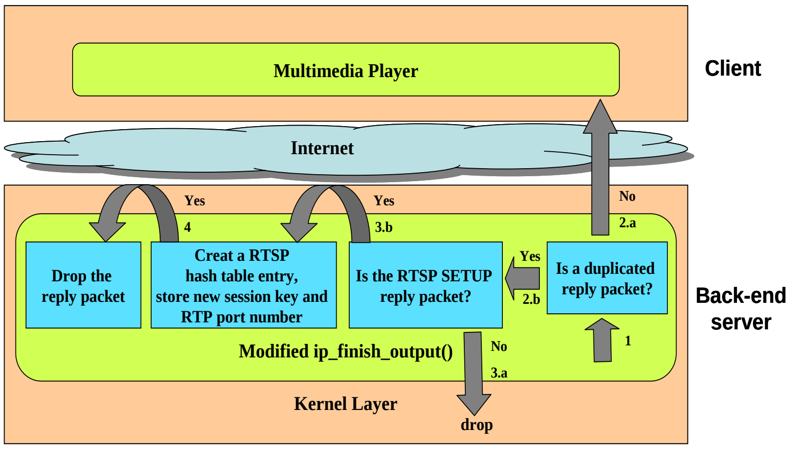
Multimedia Player
Client
Internet
Yes
4
Yes
3.b
No
2.a
Drop the reply packet
Creat a RTSP hash table entry, store new session key and RTP port number
Is the RTSP SETUP reply packet?
Yes
2.b
Is a duplicated reply packet?
Back-end server
1
Modified ip_finish_output()
No
3.a
Kernel Layer
drop

Figure 14. Packet flow in ip_finish_output() function in the back-end server

3.3.1 Handoff the Existing Streaming Service

As the back-end server initializing a RTP streaming connection to transfer the film, it will generate a corresponding initial sequence number in the first sent RTP packet. Because of the length of films are fixed, we can easily know the end sequence number of the last RTP packet. Actually, the client could get the initial number by the RTSP reply packet. Thus, the back-end server fetches the RTSP PLAY reply packet at the kernel layer, it uses the RTP port number to create the RTP hash table entry and then store initial sequence number of the RTP packet into the RTP hash table.

The back-end server uses the end sequence number of the RTP packet to determine whether it should send a notification packet to the front-end server or not. When the sequence number of the packet approaches the end sequence number of the RTP packet, the back-end server would send the notification packet to the front-end server to handoff the existing streaming connection. The content of the notification packet includes the client IP address and the RTSP port number.

4. Performance Evaluation

4.1 Benchmark and Workload of SPECweb2005

SPECweb2005 is a performance benchmark tool developed by the Standard Performance Evaluation Corporation (SPEC). It is composed of three systems: a web server, an application server, and over than one web client simulators. The benchmark is run on many clients that use port 80 to send HTTP requests to the web server. All clients are controlled by one prime client that is also a normal client. In our experiment, we construct a web cluster instead of a single server to serve clients' requests. In addition, we use the simulator servers to simulate the application servers, such as database servers.

SPECweb2005 benchmark generates a number of client connections that correspond to the setting of the number of simultaneous sessions in the configuration files. The clients will continuously send requests to web cluster system. A new user session starts as soon as the previous user session is over, and this process continues until the whole benchmark is completed.

There are three frames of SPECweb2005 workload: Banking, e-Commerce, and Support. They are designed to measure the performance of static and dynamic web services, and we selected SPECweb_Ecommerce as our workload. This workload was developed by analyzing log files as actual E-commerce sites, as well as browsing popular web stores to gather statistics such as average page size, access frequencies, and capturing form data that a customer typically fills out when purchasing products. As using the E-commerce workload in SPECweb2005, the server has to hold session information. For this purpose, we set up a cache server by Memcached software on the front-end server to share the session information with back-end servers. Therefore, our front-end server can efficiently cache the session information, such as user identify ID, items of shipping cart, and browsing history of clients in the main memory.

Because the SPECweb2005 benchmark does not provide the streaming requests, we add streaming type of requests into our workload in the experiments. We installed the DSS in our back-end servers to handle the streaming type of requests. The clients use port 554 to send streaming type of requests to our web cluster. Each client uses the VLC media player to watch the streaming films, and a new streaming request is sent to our web cluster as soon as the previous streaming service finishes.

4.2 Experimental Environment

We construct our cluster with eight back-end servers and one front-end server which acts as a cache server. Two simulator servers are used to simulate the application servers, such as database servers. In addition, ten computers are clients to make HTTP and streaming requests. All computers are connected in1Gbps Ethernet LAN by ZyXEL Dimension GS-1124 switch. The environment is shown in Table 1.

Table 1. Hardware / Software environment

	Front-end	Back-end	Simulator	Clients
CPU (Hz)	P4 3.4G			P4 2.4G P3 800M
RAM(DDR)	1G	384MB	1GB	256MB
NIC(Mbps)	Intel Pro 100/1000		D-Link DGE-530 T	Realtek RTL8139 Intel Pro100/1000 D-LinkDGE-530T
IPVS	1.21	X	X	X
Streaming Server	X	DSS 5.5.5	X	X
Media Player	X	X	X	VLC 0.9.9
Benchmark	X	X	SPEC-web2005	SPECweb2005
Num of PCs	1	8	2	10

4.3 Performance Evaluation Results

During our experiments, we found a limitation of the LVS and the TCP connection would be expired over two minutes. Therefore, we set the run-time of all films to two minutes. During the short runtime, the front-end server cannot determine whether the current streaming connection needs to be handoff if a longer film is playing. For evaluating the performance of using out proposed RTSP mechanism, we adjust the ratio of films which needs to be handed off. In the following experiments, we assumed two scenarios which are synthetic workload and heavy workload. Then we presented the results of experiments of the LVS-CVD web cluster using the RTSP Handoff mechanism and GCAP policy.

4.3.1 Synthetic Workload

In this experiment, besides the Ecommerce workload of SPECweb2005, each client only watches one streaming movie at the same time, and it will send a new streaming request to our web cluster as soon as the previous streaming service is finished. The SPECweb2005 benchmark needs about thirty minutes for running a phase of experiment. All films' runtime which a client requests is two minutes. For processing the streaming requests constantly, each client will send a total of fourteen streaming requests during one phase of experiment. Besides, for avoiding too much overhead caused by the frequent handoff of the streaming connection for our web cluster, we set the playing time of the movie which needs handoff to one minute. In other words, the old back-end server would provide the first one minute of the movie data, and the remaining one minute of the movie data is provided by the new back-end server.

Figure 15 shows the experimental results of the LVS-CAD using the RTSP Handoff mechanism with GCAP policy and different settings. The percentage means that the handoff ratio of all films. For example, "without RTSP Handoff" means the LVS-CAD web cluster provides all films without using our proposed RTSP Handoff mechanism, "50% Handoff" means that there are a half of all films which need to be handed off in the web cluster, and "100% Handoff" means the LVS-CAD web cluster provides all films with our proposed RTSP Handoff mechanism. The X-axis denotes the number of clients simultaneously issuing requests in each phase of experiment. The Y-axis denotes the total amounts of web files transferred on MBytes in each phase of experiment.

In this experiment, the web cluster using our proposed RTSP Handoff mechanism with “50% Handoff” can achieve the best performance, and outperforms the one without using RTSP Handoff mechanism by 12.85-29.55%. We also found that excessive RTSP Handoffs could degrade the performance of the whole web cluster.

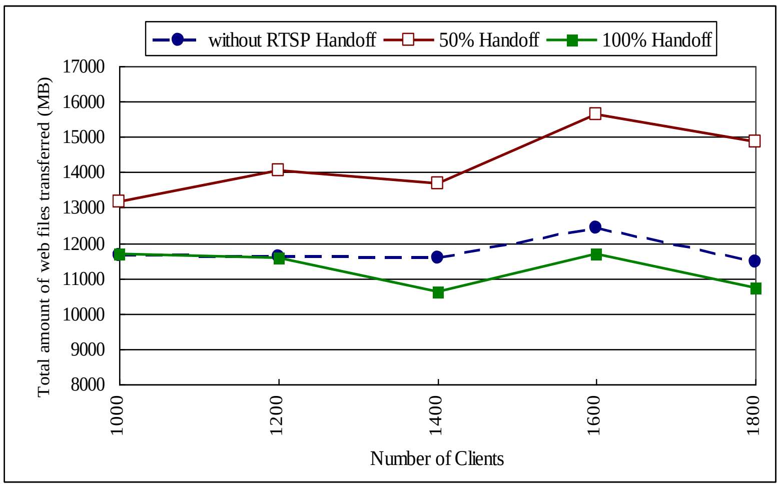

Figure 15. Performance of the LVS-LAD under synthetic workload

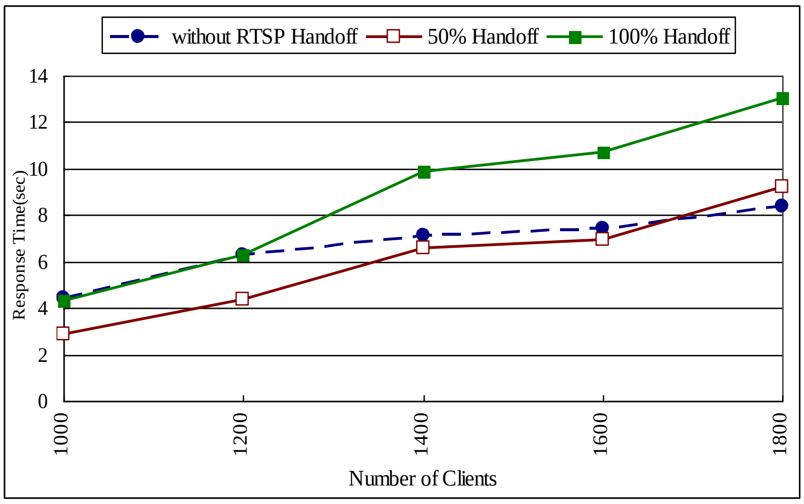

Figure 16. Average response time of LVS-CAD under synthetic workload

Figure 16 shows the average response time of LVS-CAD using the RTSP Handoff mechanism, CAP policy, and different settings. The X-axis means the number of clients simultaneously issuing requests in each phase of experiment. The Y-axis means the average response time (second) in each phase of experiment. In this experiment, the web cluster using our proposed RTSP Handoff mechanism with “50% Handoff” can obtain the smallest average response time, and outperforms the one without using RTSP Handoff mechanism by 6.94-34.59%. Excessive RTSP handoffs could also increase the average response time of requests during the experiment.

4.3.2 Heavy Streaming Workload

For demonstrating the web cluster can gain better performance using our RTSP Handoff mechanism while processing the heavy streaming workload, we increase the amount of streaming requests in the synthetic workload used in previous section in this experiment. Each client watches two streaming movies at the same time, and it will send a new streaming request to our web cluster as soon as the previous streaming service is finished. Therefore, each client will send a total of twenty-eight streaming requests during one phase of experiment. All films’ runtime which a client requests is two minutes. We set the playing time of the movie which needs handoff to be one minute.

Figure 17 shows the experimental results of the LVS-CAD using RTSP Handoff mechanism, GCAP policy, and different settings. In this experiment, there are one third of all films which need to be handed off in the web cluster. In this experiment, the web cluster using our proposed RTSP Handoff mechanism can achieve the better performance, and outperforms the one without using RTSP Handoff mechanism by 10.99-37.19%.

Figure 18 shows the average response time of LVS-CAD with the RTSP Handoff mechanism, CAP policy, and different settings. In this experiment, the web cluster using our proposed RTSP Handoff mechanism can achieve the less average response time, and outperforms the one without using RTSP Handoff mechanism by 15.54-34.35%.

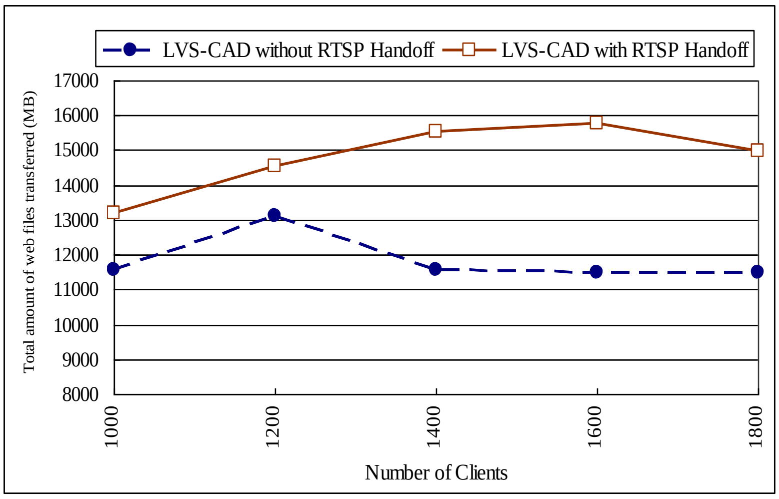

Figure 17. Performance of the LVS-CAD under heavy streaming workload

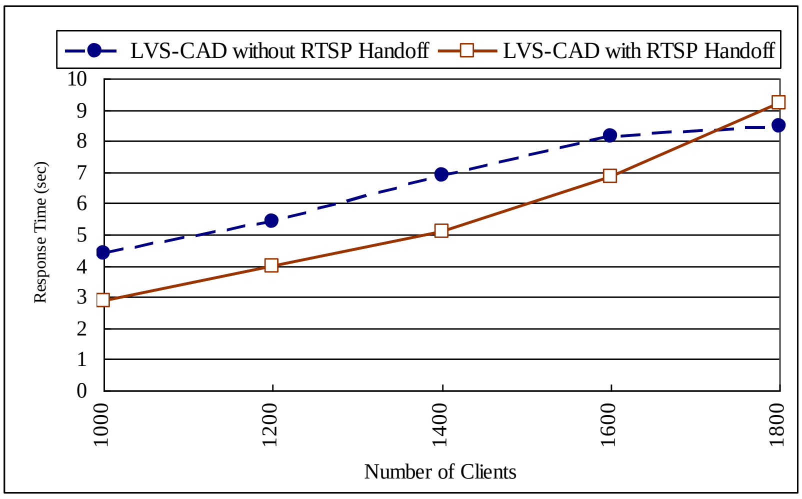

Figure 18. Average response time of LVS-CAD under heavy streaming workload

5. Conclusions and Future Work

We have designed and implemented a web cluster that can support not only static and dynamic web requests but also streaming requests. To achieve better load balance in our web cluster while providing multiple web services, we proposed and implemented the RTSP Handoff mechanism in our web cluster. Experimental results show that the LVS-CAD web cluster with our proposed RTSP Handoff mechanism can achieve 37.19% better throughput under the heavy streaming workload, and 29.55% under the synthetic workload than the one without our proposed mechanism. Besides, the average response time of the LVS-CAD web cluster when using our proposed RTSP Handoff mechanism can be reduced by 34.35% under the heavy streaming workload, and 34.59% under the synthetic workload.

We are now evaluating LVS-CAD web cluster with our proposed RTSP Handoff mechanism using more long runtime films in our experiments. Based on the LVS-CAD platform, other issues could be further explored or enhanced, such as supporting secure sockets layer, providing session affinity, and content placement and management.

References

[1] Ho-Han Liu, Mei-Ling Chiang, and Men-Chao Wu, “Efficient Support for Content-Aware Request Distribution and Persistent Connection in Web Clusters,” Software Practice & Experience, Volume 37, Issue 11, Pages 1215-1241, Sep. 2007.

[2] Mei-Ling Chiang, Yu-Chen Lin, and Lian-Feng Guo, “Design and Implementation of an Efficient Web Cluster with Content-based Request Distribution and File Caching,” Journal of Systems and Software, Vol. 81, Issue 11, pp. 2044-2058, Nov. 2008.

[3] Chun-Hung Wu, “A Kernel Mechanism for Efficiently Supporting Quality of Service in Web Cluster System”, Master Thesis, Department of Information Management, National Chi-Nan University, Nantou, Taiwan, July 2008.

[4] Wensong Zhang, “Linux Virtual Servers for Scalable Network Services,” OTTAWA Linux Symposium, July 19-22, 2000, Canada.

[5] Po-Kai Chiu, “Design and Implement a MP4 Video-on-Demand System based on PC Cluster”, Master Thesis, Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, 2003.

[6] Wei-Yuan Lin, “Design and Implementation of A PC Cluster-Based QuickTime Streaming Server”, Master Thesis, Department of Computer Science and Information Engineering, National Cheng Kung University, 2003.

[7] Emiliano Casalicchio and Michele Colajanni, “A Client-Aware Dispatching Algorithm for Web Clusters Providing Multiple services”, WWW10, May 1-5, 2001, Hong Kong.