미디어 프레임: HTTP 리디렉션을 통한 병렬 멀티미디어 시스템 구조

김 성 기[†] · 한 상 영[††]

요 약

한 대의 비디오 서버가 확장성, 용량, 고장 감내(Fault-tolerance), 비용 효율성(Cost-efficiency)에서 한계 들을 드러냄에 따라 해결책 들이 나타났다. 그러나, 이러한 해결책들도 각각의 문제점 들을 지닌다. 우리는 이러한 문제점 들을 해결하고 다양한 비디오 서버 들을 활용하기 위해, HTTP 레벨 리디렉션(Redirection)을 통하여 이질적인 퍼스널 컴퓨터(Personal Computer), 운영 체제(Operating System), 비디오 서버 들로 내용에 따른 라우팅(Routing)을 지원하는 병렬 멀티미디어 시스템 구조를 디자인하였다. 또한 프로토타입을 개발 하였으며 서로 다른 비디오 서버 제품들을 프로토타입에 추가하였고 오버헤드를 측정하였다.

키워드 : 멀티미디어 시스템, 비디오 서버, 병렬 처리, LVS, DNS, CDN

MediaFrame: Parallel multimedia system architecture through HTTP redirection

Kim SeongKi[†] · Han SangYong[††]

ABSTRACT

As a single video server exposes its limitation in scalability, capability, fault-tolerance, and cost-efficiency, solutions of this limitation emerge. However, these solutions have their own problems that will be discussed in this paper. To solve these problems and exploit various video servers, we designed a parallel multimedia system architecture that supported a content-aware routing to heterogeneous personal computer (PC), operating system (OS), video servers through a HTTP-level redirection. We also developed a prototype, added different video servers into the prototype, and measured its overheads.

Key Words : Multimedia System, Video Server, Parallel Processing, LVS, DNS, CDN

1. Introduction

The multimedia transmission has been widely used as real-time and digital broadcasting, distance learning, multimedia mail, Internet Protocol Television (IPTV) and Voice over IP network (VoIP) have significantly increased. In addition, a network capacity has increased enough to simultaneously transmit contents to various clients. With these trends, Internet video servers have widely spread.

When using a video server for multimedia transmission, if the total bandwidth used by clients exceeds the network bandwidth of the server, the server cann't transmit more contents through the overused network interface of the server. When a central processing unit (CPU) is overused, the server cann't accept more connections, process more requests or transmit more contents. In other words, network and CPU resources restrict the transmitting capacity of a video server. Using a single server, it costs a lot to extend either network or CPU capacity. A single server cann't handle a failure. For example, if the server fails, all of its services would be terminated because there are no available servers. Consequently, a single server approach has limitation in capacity, scalability, availability, fault tolerance and cost efficiency aspects.

Parallel processing technologies, which consist of video servers that provide real services and a load balancer that provides a request-routing service, can overcome this limitation. Request routing, sometimes referred to as request distribution, plays the role of distributing a client re-

† 준 회 원 : 서울대학교 전기컴퓨터공학부 박사과정
†† 정 회 원 : 서울대학교 계산통계학과 교수
논문접수 : 2006년 7월 5일, 심사완료 : 2006년 11월 2일

quest to a suitable server. Many researchers have studied request-routing mechanisms, which have been developed at various levels such as client [1], DNS [2], transport layer (Layer 4) or below [3, 4, 5, 6, 7, 8], and application layer (Layer 7) [9].

However, most request-routing mechanisms have the common weakness that the same content must be copied to all of the servers because these mechanisms cann't recognize the requested content. Although the storage capacity has increased, storing the same contents on all of the servers is the waste of storage especially for a multimedia system because the sizes of video files are huge. In order to minimize the unnecessary waste, content-aware request routing can be used. The content-aware request routing distributes different contents to each server and routes the client requests to the server with the requested content. Besides the storage waste, storage scalability can be also achieved through the content-awareness. When a new storage is added into a multimedia system and a request-routing system has the contents information in the newly added storage, the multimedia system can immediately begin streaming the contents. The content-awareness can also support session integrity, sophisticated load balance, and differentiated services [10].

Although some request routing mechanisms support the content-awareness at layer 4, it is difficult for them to be virtually used because of their limited OS support or complex processes (That will be discussed later). Layer 7 technologies don't support simultaneously various video servers because the technologies were developed for a single video server product. For example, when a Microsoft Windows Media Server is used for building a parallel multimedia system, Real network's Helix Universal Server isn't allowed to operate cooperatively with the Media Server. As available video servers, many commercial servers such as Microsoft Windows Media server, Real network's Helix Universal server, and Apple's Darwin streaming server as well as research prototypes such as Tiger [11] and SPIFFI [12] exist. These varieties made an administrator select one of video servers.

To support a content-aware routing to heterogeneous video servers, we designed a simple architecture that used the redirection through a web server. We developed a prototype and added different servers and measured the overheads.

Section 2 describes various related works by other researchers. Section 3 describes the architecture and the implementation. Section 4 describes the processes of adding heterogeneous servers into the implementation and the overheads. Section 5 concludes.

2. Related Works

This section describes the parallel processing technologies such as LVS, DNS, TCP splicing and TCP handoff in addition to the network technologies such as CDN and OpenCDN, which can be used to build a parallel multimedia system.

2.1 Linux Virtual Servers (LVS)

The Linux Virtual Server (LVS) is a software tool that supports load-balance among multiple Internet servers. LVS supports three different redirecting methods: Network address translation (NAT), IP tunneling and direct routing.

In LVS via NAT, when a load balancer receives a user request, it selects a real server, rewrites the destination address of a request IP packet to the real server IP address, which can be within a private or public network, and forwards the modified packet to the dynamically selected server. After the real server receives the packet, processes it and sends the response IP packet to the load balancer, the load balancer changes the source address of the response packet to the load balancer's address (VIP). The user then transparently receives the packet from the load balancer.

In this method, real servers can run any OS with TCP/IP, can use private IP addresses, and the load balancer needs only an IP address. However, it is limited because the load balancer bottlenecks. The performance cann't be scaled because all request and response packets pass through the load balancer.

In LVS via IP tunneling, a load balancer creates a new IP datagram that encapsulates the original datagram by using IP tunneling technology. It then forwards the new datagram to the dynamically selected server. After the real server receives the packet and decapsulates and processes it, the real server returns the replies directly to the user while maintaining the original VIP.

In this method, real servers can be on different network from the director, and a director doesn't become a bottleneck. However, the OS of a real server should support IP tunneling. Some OS need to do complex processes such as a kernel patch in order to support IP tunneling. In addition, even real servers need public IP addresses, and both a director and real servers have the additional overheads of IP encapsulation and decapsulation.

In LVS via direct routing, a load balancer adds only the Medium Access Control (MAC) address to the request data frame. It then forwards the new data frame to the dynamically selected server. After the real server receives the packet, and processes it, the real server returns the replies directly to a user.

In this method, the real servers and a director doesn't have IP tunneling overheads. However, the LVS via direct routing is limited in that the load balancer and real servers should be connected within a single physical network. In addition, the real servers should have network interface that doesn't do an Address Resolution Protocol (ARP) response so that it can avoid network IP address collision, and even real servers also need a public IP addresses.

In addition to these drawbacks, the three supported routings don't support a content-awareness, which is especially important for a multimedia system.

2.2 Domain name service (DNS)

A Round-Robin DNS extends a general DNS to provide a load balance by mapping a domain name to several IP addresses and returning one of the addresses. If a client makes a name resolution request to the Round-Robin DNS, the DNS returns an IP address in a sequential manner. Although a clustering system can be built through the Round-Robin DNS simply by editing the configuration file of DNS software, it cann't realize fault tolerance because it unconditionally returns an IP address without checking the status of a server. In addition, the intermediate DNS server between a client and the Round-Robin DNS server can cache the IP address mapped to a domain name differently from the Round-Robin DNS, which leads to fail to equally balance loads. Besides these fault-tolerance and load-balance problems, the clustering system using DNS doesn't support the content-awareness.

2.3 Other TCP layer approaches

The approaches mentioned above are in use and can be used for a cluster system. Other approaches such as TCP splicing [7] and TCP handoff [8] exist and are integrated into some research prototypes that support content-awareness at the TCP layer (Layer 4).

TCP splicing optimizes the front-end relaying algorithm by integrating the address-changing mechanism of a request and a response into a kernel. Although TCP splicing improves the front-end relaying by removing the copying and context-switching overheads, it requires modification of a front-end kernel, and the front-end system frequently bottlenecks.

In the TCP handoff mechanism, the back-end replies directly the results to a client without passing through the front-end, and the client acknowledges the reply by sending it through the front-end to the back-end. Although TCP handoff improves the performance of TCP-splicing by directly returning results and integrating the mechanism into the kernel, the kernels of both the front and back-end systems must be modified [13].

These two approaches commonly support the content-awareness at layer 4 in the kernel. However, they are only supported by limited OS such as Linux and FreeBSD [14, 15], and it is difficult to newly implement them in the kernels of various OS.

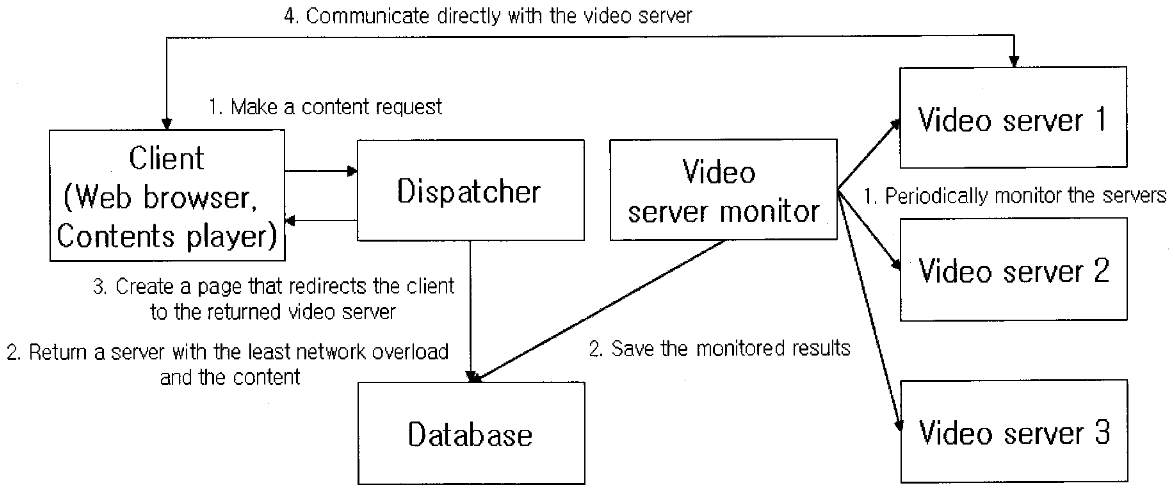
2.4 Content delivery network (CDN) and OpenCDN

CDN, a technology that was originally developed for World Wide Web (WWW), places servers (sometimes called as replicas or surrogates) and distributes contents to the servers. Clients are served by the servers according to policies such as network proximity, geographical proximity, and response time.

CDN has the sub elements such as origin servers, surrogate servers, distribution systems, request-routing systems and accounting systems. Origin servers have contents that are created by content providers and are delivered to clients or surrogate servers. Surrogate servers are servers that provide users with real services on behalf of origin servers. A distribution system distributes content to surrogate servers when the surrogate servers anticipate a client to request the contents, push, or a client makes a request to the surrogate server, pull. A request-routing system redirects a client request to a surrogate server. An accounting system records both the contents distribution to the surrogate servers and transmission activities to clients. A distribution system interacts with a request-routing system in order to notify content availability and interacts with an accounting system in order to notify the content distribution. A request-routing system interacts with a distribution system in order to notify the content demands so that the distribution system can place contents in the suitable surrogate servers. A request-routing system interacts with an accounting system in order to notify contents distribution so that the accounting system can record the distribution. An accounting system uses the collected information for a bill. [16]

OpenCDN is an open CDN implementation that supports vendor-independence and scalable delivery of live streaming content to large audience.

OpenCDN, which consists of relay nodes that perform content delivery, a portal that is the contact point of clients, and request-routing and Distribution Management (RRDM)


4. Communicate directly with the video server
1. Make a content request
Client (Web browser, Contents player)
Dispatcher
Video server monitor
Video server 1
Video server 2
Video server 3
1. Periodically monitor the servers
3. Create a page that redirects the client to the returned video server
2. Return a server with the least network overload and the content
2. Save the monitored results
Database

(Figure 1) Overall architecture and processes

that records footprint information from nodes, chooses a relay node by a client request, creates a distribution tree and dismantles the distribution tree after transmission ends. In other words, a relay node acts as a surrogate server and a RRDM acts as a request-routing and distribution system of CDN. Whenever a node boots, the node registers its capabilities and footprint information with the RRDM. Whenever a client contacts an announcement portal, the portal passes the request to the RRDM through XML-RPC [17]. The RRDM chooses the best relay, creates a distribution tree and returns the Uniform Resource Identifier (URI) to the portal. If the selected relay doesn't have the requested content, it pulls the stream from the source. The portal generates a page that redirects the client request to the returned relay node. After the requesting client finishes watching the content, the RRDM dismantles the distribution tree.

The nodes in the distribution tree are classified as FirstHop (FH), Transit (TR) and LastHop (LH) from the least to the most specific footprint. A FH relay has the widest footprint and is the first point for delivering the content. A transit relay distributes the content from the FH relay or TR relays to the other transit relays or the LH relay. The LH relay finally delivers the content to the clients [18]. RRDM doesn't have to be related to deliver contents to a client, so it reduces RRDM overheads.

OpenCDN supports vendor-independence by adding a new adaptation layer that creates a direct request to the streaming or video servers that are installed on relay nodes.

3. Architecture and Implementation

This section describes the architecture and the implementation for supporting content-awareness, load-balance and fault-tolerance as well as exploiting various servers.

3.1 Overall architecture

The overall architecture and processes are demonstrated in Figure 1.

This architecture consists of a client, a dispatcher, a video server monitor, a database, and video servers. A client is used to watch the contents, and it must have both the web browser and the suitable players according to the video servers. A dispatcher is the client's contact point and is implemented using a dynamic web page executed by a web server. A video server monitor periodically monitors video servers and saves the monitored results on the database server. A database records the server information, the monitored status of video servers, and the contents information of video servers. A video server streams contents by unicast, multicast or broadcast methods.

The service processes are summarized as follows. Whenever a client makes a content request to a dispatcher, the dispatcher retrieves the video server with both the least response time and the requested content from the database. It then retrieves the URI to the content in the video server. The dispatcher returns the status code 302 that redirects the web browser on the requesting client to the URI. After receiving the new location, the web browser automatically opens the suitable player, which makes a request to the video server located by the returned URI, and starts playing it. We adopted the least response time scheduling because network delay was considered as a more general limitation than CPU.

The architecture redirecting through a HTTP web server can be used as a CDN. A dispatcher system is similar to the CDN request-routing system in that the dispatcher system redirects a client to the suitable server. A database server is similar to the accounting system because the database records the distribution of contents and includes the transmission records. A video server can also be recognized as the surrogate server of a CDN because it provides a real service. When we use this architecture as a CDN, we have no automatic distribution system at the writing time. From this perspective, our architecture and processes can be seen as one of CDN implementations that redirect at the application level according to the given URI with the least response time scheduling.

The architecture is similar to the OpenCDN architecture because the dispatcher acts as a portal and an RRDM, and a video server plays the role of a relay node. This request routing is also similar to that of OpenCDN because both commonly operate at the application level. However, the architecture differs from the OpenCDN architecture because this architecture has no distribution system, doesn't create a distribution tree, doesn't dismantle it, and doesn't register a node to the dispatcher system.

These architecture and processes have the advantage that any video servers can be added into the clustering without modification or addition because it doesn't have a registration process, or a distribution tree, and it doesn't need to change a server setting or newly implement an adaptation layer. The added server operates cooperatively with other video servers as far as both players and a web browser with the capabilities of both HTTP connection and redirection are installed on the requesting client. It provides fault tolerance because the video server monitor monitors video servers and avoids scheduling a failed server. It also provides load balance because a dispatcher system redirects the client to the video server with the least response time. It also supports global environments because both the dispatcher and video servers don't have to be connected within a single physical network.

This architecture supports content-awareness, which allows contents to be freely distributed to any servers as far as the database records the location. The content-awareness is enabled by the database mapping the logical URI to the physical URI. Whenever a client makes requests to the dispatcher with video ID (logical ID), the dispatcher retrieves the physical URI of the video ID from the database and returns the URI to the requesting client with the 302 redirection code.

However, the architecture has more startup time overheads than low level (L4) request-routings discussed in Section 2.1, Section 2.2, and in Section 2.3. Whenever a client makes a content request to the dispatcher, the dispatcher reads information on the server status from the database server, returns the status code that makes a web browser execute the player that can watch the content, and the player makes a direct request to the scheduled server. These processes delay the time of starting the content. Another problem is that all streaming services can be stopped due to the dispatcher failure. If the dispatcher stops its service, all of new requests would be failed because the dispatcher is the only contact point of all clients at the writing time. Besides the start time overheads, and the dispatcher failure problem, the dispatcher can be bottleneck because all of the requests pass through the dispatcher. However, this bottleneck problem is minimized because the video server replies directly to the client and the dispatcher doesn't participate in the streaming process. These drawbacks are related to our goal not to minimize overheads for broadcasting but to develop a flexible architecture for the various servers and content-awareness. In addition, the dispatcher failure problem, and the bottleneck problem are common problems among all of the centralized dispatcher approaches.

3.2 Implementation

We developed the aforementioned architecture and processes to verify its practicality. We used the IIS 5.0, the Windows 2000 Server patched by service pack 4 for developing the dispatcher, the Visual Basic 6.0 patched by service pack 6 for developing the video server monitor, and the Microsoft SQL server 2000 patched by service pack 4 as a database.

3.2.1 Dispatcher

A dispatcher is implemented as a dynamic web page that uses Active Server Page (ASP). After the dispatcher retrieves the requested content and the least response time server from the database, the dispatcher generates the status code 302 that redirects the web browser on the client to the selected server. The web browser realizes that it cann't process the returned content, opens the player supporting the content, the executed player directly makes a request to the content, and starts playing the content.

〈Table 1〉 Tables for the architecture

Name	Description
CODEMAST	This table includes the classification information.
CLIENT_STAT	This table includes the client status information.
VIDEO_INFO	This table includes the video information.
VIDEO_LEVEL	This table includes the video classification information.
PLAY_INFO	This table includes the play information by clients.
FILE_INFO	This table includes the file information.
FILE_STAT	This table includes the file usage information.
SERVER_INFO	This table includes the video server information.
SERVER_STAT	This table includes the server status information.

The dispatcher could be created by using any mechanisms such as Java Server Page (JSP), Common Gateway Interface (CGI) or PHP that could access the database. However, we chose ASP because it was the basic dynamic page mechanism supported by a Windows server.

3.2.2 Video Server Monitor

A video server monitor periodically retrieves the video server information from the database, gains response times from the retrieved video servers and stores the information to the database. We used the Visual Basic 6.0 for developing the video server monitor. The monitoring time could be modified when the video server monitor starts and is basically 10 seconds in the case of no inputs.

To measure the response time, we used the IP helper functions [19] that could be accessed from the Visual Basic. The response time is measured by both the ICMP echo request and ICMP echo reply and is the same as Ping's round trip time. The usage of ICMP messages removed the necessities of developing another server daemon because most of the OS support the TCMP protocol.

3.2.3 database

A database stores the managed video servers, the URI of contents that the video servers support and the response times of video servers. The database can be from any vendors as far as it has basic mechanisms to create a table, store data into the table, retrieve them, and support the retrieving mechanism from the dispatcher.

We created the tables in Table 1 in order to realize the architecture in Section 3.1.

When a video server is added into the architecture, the information should be added into SERVER_INFO table so that the video server monitor can recognize the server. The video server monitor inserts the response time information into SERVER_STATE table. When a manager adds contents to one of video servers, contents information should be inserted into FILE_INFO and VIDEO_INFO. When inserted, the dispatcher recognizes the added contents, and clients can watch the content. The FILE_STAT, PLAY_STAT and CLIENT_STAT tables are used for statistics. This information can be used to generate a report on content usage. Both CODEMAST and VIDEO_LEVEL tables are used as reference tables for filling the VIDEO_INFO and FILE_INFO tables.

4. Results

This section describes the experience that adds the Microsoft's Windows Media Server and the Real network's Helix Universal Server 9.07 to the developed prototype. This section also shows the comparison table and the start-up overheads.

4.1 Video server

In order to add the Media Server running on the Windows 2000 server, we added its IP address into a SERVER_INFO table. A video server monitor immediately recognized the server addition and started measuring the response time by ICMP messages every 10 second and storing the result in the SERVER_STATE table. The contents information was added into the FILE_INFO and the VIDEO_INFO tables. Whenever a user contacted the dispatcher, the dispatcher retrieved the server with both the least response time and the requested content, and returned the status code 302 to the new URI. The web browser executed the suitable player that made a request to the new URI, and played the content.

We also added the Helix Universal Server 9.07 running on Solaris 5.9 as an entirely different environment following the same procedures. In this way, a video server could be any PC, Workstation, Mainframe, or OS that has the capability of running a video server. This is because the

〈Table 2〉 Problems

Problems	LVS via NAT	LVS via IP tunneling	LVS via direct routing	DNS	TCP handoff	TCP splicing	OpenCDN	HTTP redirection
Dispatcher bottleneck	○	△	△	X	○	△	△	△
Interruption overheads by dispatcher during streaming	○	△	△	X	○	△	X	X
Single network	X	X	○	X	X	X	X	X
Another network interface	X	X	○	X	X	X	X	X
Load unbalance	X	X	X	○	?	?	?	X
Fault untolerance	△	△	△	○	?	?	?	X
Additional implementation on video servers	X	X	X	X	X	X	○	X
OS modification	X	△	X	X	○	○	X	X
Content unawareness	○	○	○	○	X	X	?	X
Startup overheads	△	△	△	△	△	△	○	○
Heterogeneousness	○	○	○	○	○	○	△	X
Public IP address	X	○	○	○	X	○	○	○

SERVER_INFO table includes the IP addresses, and most of OS support the standard ICMP protocol. The other reasons are that both the VIDEO_INFO and the FILE_INFO tables include the URI that all of the video servers have, and the client is redirected through a web browser and a web server.

4.2 Client

To watch the contents, we installed both the Media Player 10.0 and RealPlayer 10.5, which was run on Windows XP patched by service pack 2. In this way, the suitable players should be installed according to the contents and the video servers.

4.3 Comparison

Before this subsection, many request routing techniques are described and HTTP redirection technique is proposed for building a parallel multimedia system. This subsection compares the described techniques for understanding.

Table 2 shows the described problems and request routing techniques till now. Dispatcher bottleneck problem means that the bottleneck can happen at the dispatcher, sometimes called as load balancer, director, front-end, or portal. Single network problem means that the dispatcher and video servers should be connected within a single physical network. Another network interface problem means that a video server should have another network interface. Load unbalance problem means that the loads of video servers mayn't be equally balanced and fault untolerance problem means that the failure of a video server can lead to the entire service failure. Additional implementation problem on video servers means that a video server requires new implementation in order to participate in the clustering. OS modification problem means that the OS of the dispatcher, or video servers should be modified. Content unawareness problem means that the distribution isn't free according to the contents. Startup overheads problem mean the overheads when starting a service. Heterogeneousness problem means that various video servers cann't operate at the same time. Public IP address means that the video servers should have public IP address. ○ means the technique has the problem, △ has the little problem, X doesn't have the problem, and ? means that it isn't exact due to data shortage.

Dispatcher bottleneck problem is common to LVS via NAT, and TCP hand off because all of the requests and responses pass through dispatcher, but only the requests pass through the dispatcher at the other techniques (LVS via IP tunneling, LVS via direct routing, TCP splicing) or the request passes through the dispatcher only when starting (OpenCDN, HTTP redirection). Interruption overheads problem is caused by the dispatcher invention during streaming processes. During streaming content, the dispatcher should change IP address (LVS via NAT, TCP handoff), MAC address (LVS via direct routing), or creates IP tunnel (LVS via IP tunneling). Fault untolerance problem is caused by not monitoring video servers. Additional implementation or some OS kernel modification is required by some techniques, which disable all OS to participate in the clustering. Content unawareness problem is common to most of the techniques because they are originally developed for a web clustering that content size is small and content duplication isn't a big problem. Startup overheads problem is caused by the initial time to change IP address (LVS via NAT, TCP handoff), MAC address (LVS via direct routing), create IP tunnel (LVS via IP tunneling), create a distribution tree (OpenCDN), or retrieve the suitable server (HTTP redirection). Heterogeneous video servers are supported by HTTP redirection, but OpenCDN requires implementing an adaptation layer.

4.4 Performance

We added the redirection through a web browser and a web server into the streaming processes, which had no choice but to increase the start-up overheads. Table 3 shows the delayed start-up times for supporting the content-awareness and heterogeneous servers with media server and media player. Video server is the Windows Media Server with Windows 2000 server, Pentium 4 2.0 Ghz, and 512 MByte RAM and client is the Media Player 10.0 with Mozilla FireFox 1.5.0.7, Windows XP, Pentium 4 1.7 Ghz, and 512 MByte RAM.

The average start-up time without any redirection was 2.3 seconds. The time reduced from 8 seconds to 1 second as the test continued and the reduction was thought to be a cache effect. The average start-up time with redirection through a web server and a web browser was 6 seconds. The approximate 4 seconds difference occurred because the client web browser contacted the dispatcher, the dispatcher retrieved the least response time server with the content from the database, and the client web browser opened the player that was able to watch the returned content.

<Table 3> Start-up times

General System	Dispatcher redirection system
8	8
4	8
3	7
2	6
1	5
1	4
1	5
1	5
1	6
1	6

5. Conclusion and Future works

As multimedia transmission increases, a single server approach exposes problems that it has limited capacity, costs a lot, and cann't handle a service failure. These problems have activated the use of the parallel processing approaches with LVS, DNS, TCP splicing, TCP handoff and OpenCDN. However, these approaches have their own problems. LVS sometimes bottlenecks, needs to pass through complex processes, and needs to connect a director and real servers within a single physical network. DNS cann't support fault-tolerance, and sometimes load-balance. These approaches commonly require the duplication of contents on all servers because these low-level approaches can only see the IP packet, so they cann't redirect a client to a suitable server according to contents. Content-awareness is especially important for a multimedia system because its content sizes are huge. Both TCP splicing and TCP handoff need to modify the kernels that aren't allowed in some OS. OpenCDN requires implementing a new adaptation layer to add a new video server. Besides this request routing problems, too many commercial products and prototypes have been developed as video servers that can be used with these aforementioned request routing approaches. These varieties caused situations that some servers are incompatible with the other servers, a user who wants to build a parallel multimedia system must select the best server product after experiments, and all clients must install the suitable player. In summary, it is difficult for already existing request routings to be used due to their own problems and the variety of video servers makes the matters worse.

The described redirecting mechanism through a web browser and a web server can support any video servers regardless of the vendors at the cost of approximate 4 seconds because all requests pass through the dispatcher that is implemented using a dynamic web page operated by a web server. It doesn't bottleneck because the dispatcher doesn't participate in transmitting contents, can easily add new servers by adding both server and content information into the database, doesn't need to connect a dispatcher and video servers within a single physical network, and supports both load balance and fault tolerance through monitoring the video servers. Our goal wasn't to minimize the overheads but to solve the heterogeneity problems without modification while, at the same time, solving the content-unawareness problem that remains unsolved by the various redirecting methods: LVS, DNS, TCP splicing, TCP handoff and OpenCDN.

To the best of our knowledge, the described architecture is the first parallel multimedia system architecture that the web browser on the client makes a request to the web server, the web server returns the 302 status code with the new URI, the web browser executes the suitable player, and the player communicates with the returned server. We are currently adding other schedulings besides the least response time scheduling, developing an intelligent distribution system according to the content popularity, and developing a new architecture that has the other dispatcher systems in order to solve the fault tolerance problem at the dispatcher system. Lastly, we believed that the described architecture was worth in that it first used HTTP redirection for content-aware routing to heterogeneous video servers.

References

[1] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson and D. Culler, "Using smart clients to build scalable services", Proceedings of the USENIX 1997 Annual Technical Conference, Jan., 1997.

[2] T. Brisco, "DNS Support for Load Balancing", http://rfc.net/rfc1794.html, Apr., 1995.

[3] "The Linux Virtual Server Project - Linux Server Cluster for Load Balancing", http://www.linuxvirtualserver.org/.

[4] W. Zhang, S. Jin and Q. Wu, "Creating linux virtual servers", Proceedings of LinuxExpo Conference, May, 1999.

[5] W. Zhang, "Linux virtual server for scalable network services", Proceedings of Ottawa Linux Symposium, Jul., 2000.

[6] P. O. Rourke, M. Keefe, "Performance evaluation of linux virtual server", Proceedings of LISA Conference, pp.79-92, Apr., 2001.

[7] A. Cohen, S. Rangarajan and H. Slye, "On the performance of TCP Splicing for URL-Aware Redirection", Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, USA, Oct., 1999.

[8] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers", Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, Oct., 1998.

[9] D. Andresen, T. Yang, V. Holmedahl and O. Ibarra, "SWEB: Towards a Scalable World Wide Web Server on Multicomputers", Proceedings of 10th IEEE International Symposium on Parallel Processing (IPPS), pp.850-856. Apr., 1996.

[10] C. S. Yang and M. Y. Luo, "Efficient support for content-based routing in web server clusters", Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Oct., 1999.

[11] W. J. Bolosky, J. S. Barrera III, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold and R. F. Rashid, "The Tiger Video Fileserver", Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Zushi, Japan, pp.97-104, Apr., 1996.

[12] C. S. Freedman, D. J. DeWit, "The SPIFFI scalable video-on-demand system", Proceedings of the 1995 ACM SIGMOD international conference on Management of data, San Jose, CA, USA, pp.352-363, May, 1995.

[13] M. Aron, D. Sanders, P. Druschel and W. Zwaenepoel, "Scalable Content-aware Request Distribution in Cluster-based Network Servers", Proceedings of the USENIX 2000 Annual Technical Conference, San Diego, CA, USA, Jun., 2000.

[14] "TCPHA project", http://dragon.linux-vs.org/~dragonfly/htm/tcpha.htm.

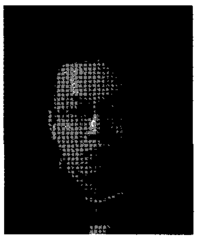
[15] "TCPSP Software", http://www.linuxvirtualserver.org/software/tcpsp/index.html.

[16] M. H. Kabir, E. G. Manning and G. C. Shoja, "Request-routing trends and techniques in content distribution network", Proceedings of ICCIT, Dhaka, Bangladesh, pp.315-320, Dec., 2002.

[17] "XML-RPC Specification", http://www.xmlrpc.com/spec.

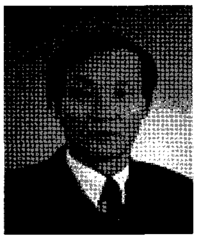
[18] A. Falaschi, "Open Content Delivery Network Short Overview", http://labtel.ing.uniroma1.it/opencdn/tnc2004.pdf, 29 Jun., 2004.

[19] Microsoft, "IP Helper Functions", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iphlp/iphlp/ip_helper_functions.asp.



김 성 기

e-mail : ditoman@chollian.net
2000년 건국대학교 컴퓨터공학과(학사)
2002년~2007년 서울대학교 전기컴퓨터 공학부 박사과정
관심분야 : 웹, 멀티미디어, 병렬 처리, 보안 등



한 상 영

e-mail : syhan@pplab.snu.ac.kr
1972년 서울대학교 공과대학 응용수학과 (학사)
1977년 서울대학교 자연과학대학 전산학 (석사)
1983년 미국 텍사스대학교 (오스틴) 전산학(박사)
1977년~1978년 울산대학교 공과대학 전임강사
1984년~1988년 서울대학교 계산통계학과 조교수
1988년~1993년 서울대학교 계산통계학과 부교수
1993년~현재 서울대학교 계산통계학과 교수
관심분야 : 병렬 처리, 보안