THE GLOBALLY SCALABLE ARCHITECTURE FOR HETEROGENEOUS VIDEO SERVERS WITHOUT ANY MODIFICATION

SeongKi Kim, SangYong Han
School of Computer Science and Engineering, Seoul National University, 56-1 Shinlim, Kwanak, Seoul, Korea

Keywords: Multimedia system, LVS, DNS, CDN, Video server.

Abstract: As a single video server reveals its limitations in the aspects of scalability, capability, fault-tolerance, and cost-efficiency, a part of solutions for these limitations emerge. However, these solutions have their own problems. To overcome these problems and exploit already existing video servers, we designed the globally scalable architecture that supports both heterogeneous and off-the-shelf personal computer (PC), operating system (OS), video server applications with both a HTTP-level redirection and the least response time scheduling without any modification or addition as well as developed its prototype.

1 INTRODUCTION

The multimedia transmission spreads widely as the demands of a real-time broadcast, a digital broadcast, distance learning, a multimedia conferencing, a multimedia mail, *Internet Protocol Television (IPTV)* (IPTV world forum, 2006), and *Voice over IP network (VoIP)* (Internet Telephony: Voice Over Internet Protocol) significantly increase. However, traditional storages such as a CD-ROM and a hard disk for content distribution can't support requirements that contents could be transmitted to many users at the real time from a device or a file, and watched by many users at the same time. These drawbacks of traditional file systems for the content distribution make video servers connected by Internet proliferate as content storage and transmission media. In addition, video servers and Internet are utilized for a multimedia transmission as a network capacity increases enough to transmit simultaneously content to various clients.

A video server supports *broadcast* or *unicast* service, sometimes referred to as *video on demand (VOD)* or *movie on demand (MOD)*, and guarantees that the appropriate amount of data is delivered to a client buffer in order to minimize a jitter or an interruption. A broadcast service transmits live contents from a device or a file to the connected clients at the real time, which can't support VCR-like interactivity such as a pause, a rewind, and a forward. A unicast service transmits pre-created contents to the connected clients at the same time, which can support VCR-like interactivity such as a pause, a rewind, and a forward. When content is transmitted as broadcast or unicast, the content occupies the constant network bandwidth usually related with a video quality. If a total bandwidth used by clients exceeds the entire network bandwidth of the transmitting server, the server can't transmit more contents through the fully used network interface. When a *central processing unit (CPU)* is overused by various services, the CPU can't accept more connections, process more requests, and transmit more contents. In other words, network and CPU resources place a restriction on the transmitting capability of a server. As a solution to these processing and bandwidth limitations, a single server approach is expensive to extend its capability. In addition, it can't handle a service failure. In summary, a single server approach for multimedia transmission has limitations in the aspects of capability, scalability, availability, fault tolerance, and cost efficiency.

To overcome these limitations, parallel processing technologies such as clustering with a *round-robin domain name service (DNS)* (T. Brisco, 1995), clustering with a *linux virtual server (LVS)* (Whensong Zhang et al., 1999), (Whensong Zhang, 2000) and (Patrick O' Rourke and Mike Keefe, 2001) , and clustering with a *OpenCDN* (Alessandro Falaschi, 2004) can be used for building a parallel multimedia system. Among these technologies, DNS maps a domain name to several real IP addresses, and answers a name resolution request of a client according to a pre-determined policy such as a round robin (T. Brisco, 1995). However, DNS approach can't provide the fault-tolerance to a client because it unconditionally returns one of IP addresses mapped to a domain name without checking the server status, and 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 designated DNS, which leads to fail to equally balance loads among nodes in a cluster. LVS can be also used for a video server clustering and is organized as a cluster of systems providing real services and a load balancer, sometimes referred to as a director or a dispatcher, operating as a contact point with a single *virtual IP address (VIP)* as well as redirecting a request to one of real servers according to policies using different redirecting methods. LVS supports 3 methods such as *network address translation (NAT)*, *IP tunneling*, and *direct routing* as redirecting methods. When NAT is chosen, a load balancer changes the destination address of a request IP packet to the real server IP address, and changes the source address of a response IP packet to the load balancer address (VIP) before returning the results to a user. NAT has the problem that the load balancer becomes a bottleneck because all of the request and the response packets pass through the load balancer that changes the destination and the source addresses. When IP tunneling is chosen as a LVS redirecting method, the load balancer creates a new IP datagram forwarded to a real server encapsulating the original datagram, and the real server directly returns the results to the requesting user after decapsulating and processing the new datagram. IP tunneling has the problems that OS of a real server should support IP tunneling, a real server should have a public address, and both a load balancer and real servers have additional IP tunneling overheads. Although most of the modern OS support IP tunneling technology, some kinds of legacy OS still exist, and some OS should do the complex processes such as a kernel patch in order to support IP tunneling. When direct routing is chosen, the load balancer changes the *Medium access control (MAC)* address of a request data frame to the destination MAC address of the selected server, and the real server directly returns the results to the requesting user. Direct routing has the problems that a real server requires a non-ARPing network interface, both a load balancer and real servers should be connected within a single physical network, and a real server needs a public address. Besides its own problems of each technology mentioned above, all of these redirecting methods at the low level have the common weakness that the same content should be copied into all of the servers because a low level redirection can't recognize the requested content but only the IP packet. Although storage capacity largely increased during the past, storing all of the same contents in all of the servers is wasteful when considering the increasing sizes of video contents. OpenCDN has an application layer request routing, and can support various servers, but it requires the implementation of a new adaptation layer that communicates with a streaming server whenever a server product is added.

As available video server systems, many commercial products such as *Microsoft's Windows Media server* (http://www.microsoft.com/windows/windowsmedia/default.aspx), *Real network's helix universal server* (http://www.realnetworks.com/products/media_delivery.html), and *Apple's Darwin streaming server* (http://developer.apple.com/darwin/projects/streaming/) as well as many research prototypes such as *Tiger* (William J. Bolosky et al., 1996) and *SPIFFI* (Craig S. Freedman and David J. DeWit., 1995) exist and are operating in the real world with their own proprietary applications, the proprietary technologies, advantages, and disadvantages. These varieties caused the current congestion that incompatible servers are operating at the same time and a practitioner should select one of them after comparing them for a long time. In this situation where servers are various, the technology that supports all of these heterogeneous servers without a dependency on any vendors and content-awareness is required so that a practitioner can freely choose a server and extend the capability.

To complement the drawbacks of each technology mentioned above and exploit already existing servers, we designed a simple, and scalable architecture that can be used within a *wide area network (WAN)*, and support content-awareness because the architecture exploits HTTP facilities as a redirecting method. Our architecture can also provide a little fault-tolerance through monitoring video servers and avoiding assigning the repetitively failed server. To prove its practicality, we also developed its prototype.

This paper is organized as follows. Section 2 refers to various related works by other researchers. Section 3 describes our architecture and implementation, and Section 4 concludes.

2 RELATED WORKS

This section describes the parallel processing technologies such as LVS, and DNS as well as network technologies such as CDN and OpenCDN.

2.1 Linux virtual server (LVS) (Whensong Zhang et al., 1999), (Whensong Zhang., 2000) and (Patrick O' Rourke and Mike Keefe, 2001)

Linux virtual server (LVS) is a software tool that supports load-balance among multiple Internet servers. LVS supports 3 different methods as redirecting methods, Network address translation (NAT), IP tunneling, and lastly direct routing.

LVS via NAT operates as shown in Figure 1.

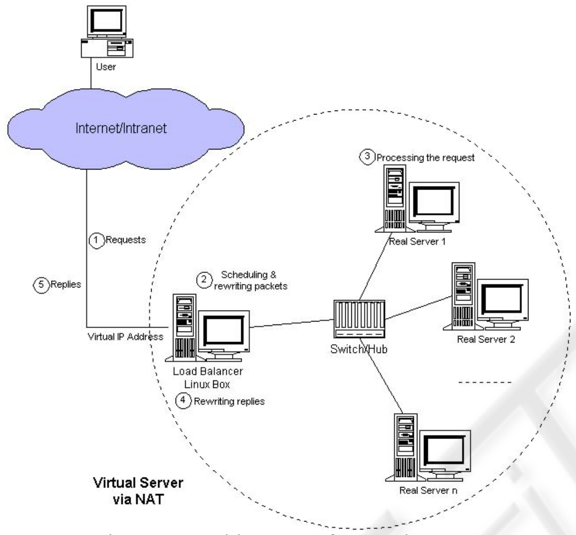

Figure 1: Architecture of LVS via NAT.

When a user makes requests to a load balancer, which selects a real server, rewrites the destination address of a request IP packet to the real server IP address that can be within a private network or a 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 requesting user through the load balancer that rewrites the replies from the real server changing the source address of a response packet to the load balancer address (VIP). The requesting user transparently receives the packet from the load balancer.

Although real servers can run any OS with TCP/IP facilities, can use private IP addresses, and only a load balancer needs public IP address in LVS via NAT, it has the limitations that the load balancer becomes a bottleneck and the performance is not scalable because all of the request and the response packets pass through the load balancer.

Figure 2 shows the architecture of LVS via IP tunneling.

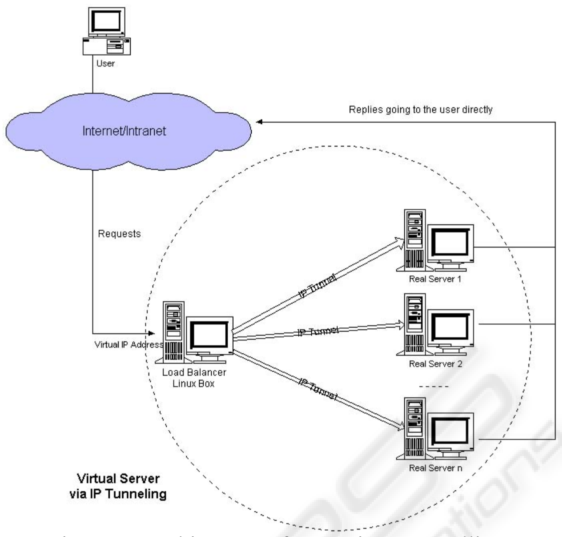

Figure 2: Architecture of LVS via IP tunnelling.

When a user makes requests to a load balancer, which selects a real server, creates a new IP datagram encapsulating the original datagram using IP tunneling technology, and forwards the new datagram to the dynamically selected server. After the real server receives the packet, decapsulates it, processes it, the real server returns the replies directly to a user.

Although real servers can be on different network from the network of a director, return directly to clients, and a director doesn't become a bottleneck, IP tunneling has the limitations that OS of a real server should support IP tunneling, some OS should do the complex processes such as a kernel patch in order to support IP tunneling, real servers also need public IP addresses, and both a director and real servers have the additional overheads of IP encapsulation and decapsulation.

Figure 3 shows the architecture of LVS via direct routing.

When a user makes requests to a load balancer, which selects a real server, rewrites only the MAC address of the request data frame to the destination MAC address, and forwards the new data frame to the dynamically selected server. After the real server receives the packet, processes it, the real server returns the replies directly to a user.

Although real servers return directly to clients, a director doesn't become a bottleneck, and both real servers and a director doesn't have IP tunneling overheads, LVS via direct routing has the limitations that the load balancer and real servers should be connected within a single physical network, the real servers should have network interface that doesn't do ARP response in order to avoid network IP address collision, and even real servers need a public IP addresses.

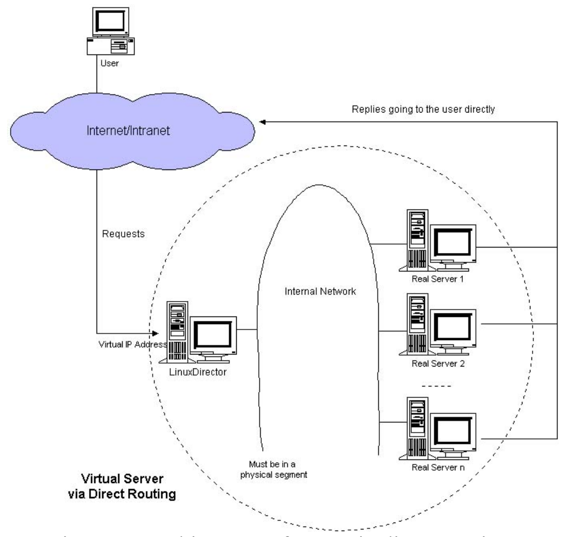

Figure 3: Architecture of LVS via direct routing.

LVS supports 4 scheduling algorithms such as Round-Robin scheduling, Weighted Round-Robin scheduling, Least-Connection scheduling, and Weighted Least-Connection scheduling. Round-Robin scheduling selects a real server in a round-robin manner regardless of the processing capacities or the number of connected clients. Weighted Round-Robin scheduling assigns a weight to a server according to its processing capacities. In the weighted round-robin, a server having more processing capacities receives a higher weight, which leads to serve more clients. Least-Connection scheduling selects a real server having the least number of connections. Weighted Least-Connection scheduling assigns a weight to a server according to the processing capacities, and tries to maintain the ratio of the active connections among the servers to that of weights. These 4 scheduling algorithms can be associatively used with 3 redirecting algorithms mentioned above.

2.2 Content Delivery Network (CDN) & OpenCDN (Md. Huamyun Kabir et al., 2002) and (Alessandro Falaschi, 2004)

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

CDN has the subelements such as *origin servers*, several *surrogate servers*, *distribution systems*, *request-routing systems*, and *accounting systems*. Origin servers have contents that are firstly created by content providers, and provide clients with broadcast or unicast service under the status owned by a content provider. Surrogate servers are the servers that provide real services to users on behalf of origin servers. A distribution system distributes content to surrogate servers when the surrogate servers anticipate a client to request the content, 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 measures and records both the content distribution and transmission activities. A distribution system interacts with a request-routing system in order to inform content availability and an accounting system in order to inform the content distribution activities. A request-routing system informs the demands of content to a distribution system so that a distribution system can place content to the appropriate surrogate servers, and the distribution of content to an accounting system so that the accounting system can record the distribution. An accounting system uses the collected information for various purposes such as charging, and statistics. (Md. Huamyun Kabir et al., 2002)

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

OpenCDN operates as shown in Figure 4.

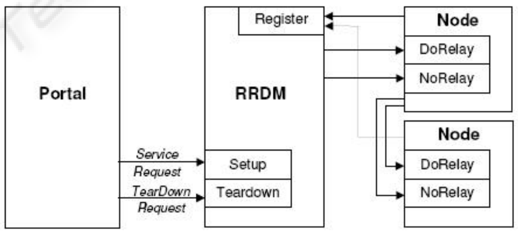

Figure 4: OpenCDN entities and their main interfaces (Alessandro Falaschi, 2004).

OpenCDN is made of a *Relay Nodes* that perform content delivery and distribution, *Portal* that is the contact point of clients, and *Request Routing and Distribution Management (RRDM)* that records footprint information from nodes, chooses a relay by a client request, creates a distribution tree, and dismantles the distribution tree after a transmission ends. In other words, a relay node plays the role of a surrogate server and a RRDM plays the roles of request-routing system and distribution system in the CDN concept.

Whenever a node boots, the node registers its capabilities and footprint information with the RRDM. Whenever a client with viewers contacts to an announcement portal, the portal makes a request to the RRDM through XML-RPC (http://www.xmlrpc.com/spec), RRDM chooses the best relay, creates distribution tree, and returns the content *Uniform Resource Identifier (URI)* (T. Berners-Lee et al., 2005) 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. After the requesting client finish 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 less specific footprint to the more specific footprint. A FirstHop relay has the widest footprint and is the first point for distributing content. A transit relay distributes the content from the FirstHop relay or the other transit relays to the other transit relays or LastHop relay. The LastHop relay delivers finally the content to the clients (Alessandro Falaschi, 2004). In order to deliver content to a client, RRDM doesn't have to be related, which reduces the RRDM overheads.

OpenCDN supports the vendor-independence through adding a new adaptation layer that makes a direct request to the streaming or video servers.

3 ARCHITECTURE AND IMPLEMENTATION

This section describes our architecture and implementation in order to support scalability, content-awareness, load-balance, fault-tolerance, and use off-the-shelf parts.

3.1 Overall Architecture

The overall architecture and process are demonstrated in Figure 5.

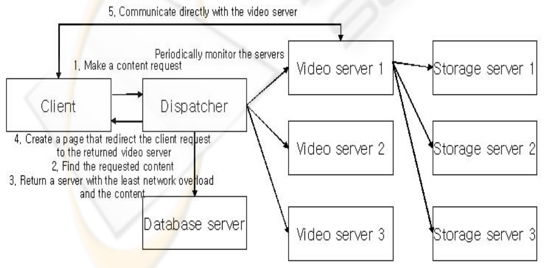
5, Communicate directly with the video server
Periodically monitor the servers
1, Make a content request
Client
Dispatcher
Video server 1
Video server 2
Video server 3
Storage server 1
Storage server 2
Storage server 3
Database server
4, Create a page that redirect the client request to the returned video server
2, Find the requested content
3, Return a server with the least network overload and the content

Figure 5: Overall architecture and process.

Our architecture consists of a client, a dispatcher, a database server, video servers, and storage servers. A client is the system that is used for watching the content by a user and it should have the proprietary players for the contents served by video servers. A dispatcher is the contact point of clients, periodically monitors video servers, and saves the monitored results in the database server. A database server stores the server information, the status of video servers, and content information. A video server takes responsible for streaming contents through unicast, multicast, and broadcast methods. A storage server has contents, and provides the contents to the selected video server according to the client requests through storage sharing mechanisms such as a network drive and *a network file system (NFS)* (http://www.ietf.org/rfc/rfc1094.txt).

The service procedures are summarized as follows. Whenever a client makes a content request to the dispatcher through *hyper text transfer protocol (HTTP)* (http://www.w3.org/hypertext/WWW/Protocols/), the dispatcher retrieves the video server with both the least response time and the requested content from the database server, and retrieves the URI (http://www.ietf.org/rfc/rfc1094.txt) to the content in the video server. The dispatcher returns a status code 302 that redirects the requesting client to the returned content location. After receiving the new location, the proprietary player installed on the client makes a direct request to the returned location, and starts playing it.

We adopted the least response time scheduling because other schedulings are not suitable for a multimedia system in that audio content and video content occupy a largely different bandwidth from web content. In other words, the server with the most number of connections is not always the most congested server in a multimedia system because the server with a few viewers who watch high-quality video contents can practically use more network bandwidth than the server with a lot of viewers who watch low-quality video contents.

Our architecture redirecting at the HTTP level can be used as a CDN. A dispatcher is similar to the request routing system of CDN in that the dispatcher makes a client redirect to the appropriate server, a database server is similar to the accounting system in that the database server records the distribution of contents and includes the transmission records. A video server can be recognized as a surrogate server in that it provides a real service. When we use our system as a CDN, we have no automatic distribution system. In this point of view, our architecture and process can be seen as one of CDN implementations redirecting at the application level according to the given URI and having the least response time scheduling.

Our architecture is similar to the OpenCDN architecture in that the dispatcher plays the roles of a portal and a RRDM, and a video server plays the role of a node. Our request routing is also similar to that of OpenCDN in that two of them are commonly operating at the application layer. However, our architecture is different from the OpenCDN in that our architecture has no distribution system, creates and dismantles no distribution tree before or after the delivery, and has no registration process of a node with the request-routing system.

These architecture and processes have the advantages in that any video servers can be added into a cluster as far as client softwares for the streaming, and the HTTP web browsers with the capabilities of both HTTP connection and redirection are installed on the requesting client. This architecture is simple to implement and add a server because this architecture has neither automatic registration process nor distribution tree, and doesn't have to change a server setting or newly implement an adaptation layer according to a server product. Our architecture supports content-awareness, which allows contents to be freely distributed into any servers as far as a database server records the location. It provides fault tolerance because a dispatcher monitors video servers, and avoids scheduling a failed server as well as provides load balance because a dispatcher makes a client redirect to the video server with the least response time. It can support global environments such as CDN because both a dispatcher and video servers don't have to be connected within a single physical network. It has only the page-generating overheads at the dispatcher because a video server directly communicates with clients.

However, our architecture has more overheads than low level (L4) request routings that are supported by DNS, LVS in Section 2.1, and use TCP splicing (Ariel Chhen et al., 1999), TCP handoff (Vivek S. Pai et al., 1998) because a dispatcher reads information about the server status from a database server, returns a status code that makes a client redirect, and a client makes a direct request to the scheduled server. In our architecture, a real server should have a public IP address in order to directly communicate with clients. All of these drawbacks are related with our goal that is not to minimize overheads for broadcast but to develop flexible architecture for supporting many existing servers and content-awareness.

3.2 Implementation

We developed the architecture and process described in Section 3.1 in order to verify its practicality. We used a *RealPlayer 10.5* running on *Windows XP* patched by service pack 2 for a client, *IIS 5.0* running on *Windows 2000 Server* patched by service pack 4, *Visual Basic 6.0* patched by service pack 6 for a dispatcher, *Microsoft SQL server 2000* patched by service pack 4 for a database server, and *Solaris 5.8* with *Real networks' Helix server 9.07* for a video server and a storage server.

3.2.1 Client

A client has the RealPlayer 10.5 that plays the content directly returned from a video server. A RealPlayer is chosen because it can be run with the Real networks' Helix universal server 9.07. If the other video servers are used, the player should be differently chosen. For example, if we add the *Microsoft's Windows Media Server* into a video server cluster, the player for the server, *MediaPlayer*, should be installed for watching its contents. In this way, the proprietary players should be installed according to the content type and the video servers.

3.2.2 Dispatcher

A dispatcher system has a video server monitor, and a dynamic web page using *Active Server Page (ASP)*. A video server monitor periodically retrieves the video server information from the database server, gains information about their availability from the retrieved video servers, and re-stores the gained availability information to the database server. We used a Visual Basic 6.0 for the video server monitor. After the dynamic web page retrieves contents, and availability information from the database server, the dynamic web page generates a status code 302 that makes a client redirect to the least response time server with the requested content. To measure the response time, we used the open *ping component* (http://www.activexperts.com/activsocket/) that is able to be accessed from Visual Basic.

The dynamic web page could be created by using any dynamic page mechanisms such as *Java Server Page (JSP)*, *Common Gateway Interface (CGI)*, and *PHP* that can access a database, but we chose ASP because it was the basic dynamic page mechanism supported by Windows server. The scheduling method used by the dispatcher system can be easily extended or modified because a dynamic web page includes a scheduling algorithm.

Whenever a client makes a request to the dynamic web page through general web browsers such as *Internet Explorer*, the browsers receive a status code 302 that redirects the requesting client to the least response time server with the requested content, and opens its proprietary player according to the content.

3.2.3 Database Server

A database server stores the managed video server information, the URI of contents that video servers provide, and the response time of video servers. It can be any database products with the basic mechanisms that can create a table, store data into it, retrieve data from it, and support the retrieving mechanism from a dynamic web page.

We chose Microsoft's SQL server 2000, and created the following tables in Figure 6 in order to realize the architecture in Section 3.1.

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_STAT	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 server information.
SERVER_STATE	This table includes the server status information.

Figure 6: Tables for the architecture.

When a server is added to the architecture, the information should be added to SERVER_INFO table so that the video server monitor operating at the dispatcher can know the server. The video server monitor fills the response time information into SERVER_STATE table. Information should be inserted into FILE_INFO and VIDEO_INFO when a manager added contents to one of video servers. If inserted, the dynamic web page at the dispatcher knows the added content and can be generated including the content. FILE_STAT, PLAY_STAT, and CLIENT_STAT tables are used for play statistics. These information can be used to generate a report about content usage statistics. Both CODEMAST and VIDEO_LEVEL tables are used as referenced tables for filling the VIDEO_INFO and FILE_INFO tables.

3.2.4 Video Server

A video server can be any PC, Workstation, Mainframe, OS with the capability of running a video server because all of the video servers have the URI pointing to the provided content, and both the VIDEO_INFO and FILE_INFO tables include the URI.

3.2.5 Storage Server

The storage servers of our architecture can be any computers, OS, or storage systems as far as they can provide the video servers with contents using storage sharing mechanisms such as a network drive and a NFS. This can be realized because storage sharing mechanisms automatically mount a disk according to the old setting when starting, video servers automatically publish the contents from any disks when starting, and a database server stores all of the publishing point information regardless of the local storage of a video server.

4 CONCLUSION AND FUTURE WORKS

As multimedia transmission significantly increases, a single server approach reveals the problems that it has limited capacity, is expensive to improve a performance, and can't handle a failure. These limitations make the parallel processing approaches with LVS, DNS, and OpenCDN proliferate. However, as available video server products for these approaches, many commercial products such as Microsoft's Windows Media server, Real network's Helix server, and Apple's Darwin streaming server as well as many research prototypes such as Tiger, and SPIFFI have been developed and have operated in the real world. These varieties caused current situations that incompatible servers are separately operating and a practitioner who wants to build a multimedia system should select one of the products after comparing them for a long time.

As a redirecting technology, we developed a HTTP-level mechanism because many low level mechanisms used by LVS, and DNS, as well as a high level mechanism used by OpenCDN have their own problems. LVS sometimes has bottleneck, needs to do complex processes, and needs to connect a director and real servers within a single physical network. DNS can't provide fault-tolerance and sometimes load-balance. Commonly, all of contents should be copied to all of the servers because the low level mechanisms can see only the IP packet, and can't redirect a client to a server according to the requested content. OpenCDN requires a new adaptation layer to be implemented for adding a new video server. Our HTTP-level redirecting mechanism can support any video servers regardless of the vendors, has no bottleneck, can easily add new servers by adding information into a database server, needs to be connected within a single physical network, which makes our architecture available for CDN, and provides both load-balance and a little fault-tolerance at the cost of more overheads caused by the HTTP-level redirecting method. Our goal was not to minimize the redirecting overheads but to solve the heterogeneity problems of the various redirecting methods supported by LVS, DNS, and OpenCDN without any modification. As a scheduling algorithm, we developed the least response time scheduling because the other algorithms are not suitable for a video server cluster in that they don't consider the difference among the sizes of content bandwidth.

The developed architecture and process have the contributions that to the best of our knowledge, they are the first architecture redirecting a client request at the HTTP-level for video servers without any modification or addition. We are currently measuring the overheads of the developed prototype, adding the other schedulings besides the least response time scheduling, and developing the intelligent distribution system according to the content popularity.

REFERENCES

IPTV world forum 2006. http://www.iptv-forum.com/2006/component/option,com frontpage/Itemid,1/.

Internet Telephony: Voice Over Internet Protocol (IP). http://engr.smu.edu/ venkatra/VoIPHTML.html.

T. Brisco. April 1995. DNS Support for Load Balancing. http://rfc.net/rfc1794.html.

Whensong Zhang, Shiyao Jin, Quanyuan Wu. May 1999. Creating linux virtual servers. In Proceedings of LinuxExpo Conference.

Whensong Zhang. July 2000. Linux virtual server for scalable network services. In Proceedings of Ottawa Linux Symposium.

Patrick O' Rourke, Mike Keefe. April 2001. Performance evaluation of linux virtual server. In Proceedings of LISA Conference. pages 79–92.

Alessandro Falaschi. 29 June 2004. Open Content Delivery Network Short Overview.

Microsoft Windows Media Homepage. http://www.microsoft.com/windows/windowsmedia/default.aspx.

Real networks Media Servers. http://www.realnetworks.com/products/media_delivery.html.

Darwin Streaming Server. http://developer.apple.com/darwin/projects/streaming/.

William J. Bolosky, Joseph S. Barrera, III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold, Richard F. Rashid. April 1996. The TIGER video fileserver. In Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV). pages 97–104. Zushi, Japan.

Craig S. Freedman, David J. DeWit. May 1995. The SPIFFI scalable video-on-demand system. In Proceedings of ACM SIGMOD. pages 352–363. San Jose, CA, USA.

Md. Huamyun Kabir, Eric G. Manning, Gholamali C. Shoja. December 2002. Request-routing trends and techniques in content distribution network. In Proceedings of ICCIT. pages 315–320. Dhaka, Bangladesh.

XML-RPC Specification. http://www.xmlrpc.com/spec.

T. Berners-Lee, R. Fielding, L. Masinter. January 2005. Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc3986.txt.

Sun Microsystems, Inc. March 1989. NFS: Network File System Protocol Specification. http://www.ietf.org/rfc/rfc1094.txt.

HTTP - Hypertext Transfer Protocol: A protocol for networked information. June 26, 1995. http://www.w3.org/hypertext/WWW/Protocols/.

Ariel Chhen, Sampath Rangarajan, Hamilton Slye. October 1999. On the performance of TCP Splicing for URL-Aware Redirection. In Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems. Boulder, CO.

Vivek S. Pai, Mohit Aron, Gaurav Banga. October 1998. Locality-Aware Request Distribution in Cluster-based Network Servers. In Proceedings of the 8th Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA.

ActiveXperts software. Network Toolkit based on WinSock. http://www.activexperts.com/activsocket/.