Scalable Video Delivery on The Web

B. Prabhakaran **Yu-Guang Tu** **Yin Wu** *
School of Computing,
National University of Singapore, Singapore 117543
Email: prabha, tuyuguan, wuyin @ comp.nus.edu.sg

June 10, 2000

Abstract

In this paper, we present a scalable video delivery application on the web. This application makes use of an object-level scalable web framework for handling large multimedia objects such as digitized movies or multimedia lectures. This scalable framework automatically monitors access patterns, replicates, and maintains large multimedia objects among the servers without the need for full URL replication. The framework employs a *traceable HTTP redirection* approach to avoid cyclic redirections among distributed web-based video servers.

We also discuss two possible routing mechanisms for scalable video delivery application: one using the server redirection facility provided by the object-level scalable web framework. The other one is using a client selection approach where the client can choose a server that offers minimum waiting time (based on servers' admission control mechanism). Routings of movie requests are done not only in the beginning of a movie presentation but also while handling dynamic user interaction such as fast forward or navigate in time.

*Supported in part by National University of Singapore Academic Research Fund Grant RP981669.

1 Introduction

Providing scalable video service in a transparent manner to clients is an interesting research issue. This scalable service should be provided not only at the time when clients launch the video player, but also during the procedure of presentation. For example, when a movie is presented, VCR-style presentation controls are often provided to enable the user to control playback such as pausing, navigating in time, fast-forwarding and rewinding the movie. If the user selects fast-forwarding or rewinding to jump to different portions of the video, and if these portions are not buffered, a new selection request will be sent to the server to request presentation from a specific frame. At that time, another server, if it is better, may be chosen to provide the movie presentation service.

In this paper, we present a scalable video delivery application on the web. This application employs an object-level scalable web framework (as opposed to URL-level replication). Main features of the scalable video delivery application are as follows.

1. It uses a scalable framework that monitors object-level access patterns, replicates, and maintains large multimedia data among servers that can be distributed in a Wide Area Network.

2. This scalable framework uses, apart from a first level DNS based redirection, HTTP redirections to handle requests based on server workloads and the geographical area from which requests are coming in. HTTP redirections among geographically distributed servers may become cyclic when servers are highly loaded. Hence, we have designed a *traceable request redirection* approach that uses *redirection history* for carrying out HTTP redirections.

3. The video delivery application can use one of the following routing mechanisms:

   - Server redirection that is provided by the object-level scalable web framework.
   - Client selection approach where a client selects a server that offers minimum waiting time for a video request.

This scalable video delivery application chooses a server not only at the beginning of a movie presentation but also when a new selection request needs to be issued for handling dynamic user requests such as navigate in time or fast forward.

2 Object-level Scalable Web Framework

The video delivery application uses an object-level scalable framework that can monitor access patterns, replicate, and maintain large multimedia data among servers. These servers can be distributed in a Wide Area Network. This object-level scalable framework incorporates a replication mechanism that monitors access patterns at the object level (movies, lecture presentations, product catalogues, and other large multimedia objects). It identifies hot objects that cause increase in server workload and replicates them in some selected servers based on their workload as well as geographical nature of access patterns. For instance, a movie that is normally delivered by a server in the US may be replicated in a server in Europe based on access patterns. This object-level scalability necessitates object-to-server(s) mapping information to be maintained and exchanged among participating servers.

The object-level scalability in our framework is dynamic in the following sense. Identification of objects to be replicated is automatically done based on access patterns. This set of hot objects gets dynamically updated too based on access patterns. For instance, less popular movies may be removed from replicated servers to make way for the more popular ones. An object in the scalable framework may be served by a set of *primary* and *secondary servers*. Primary servers are the origin web servers for this object and have the entry in the authoritative DNS servers for the corresponding URLs. One of the IP addresses of the primary servers will be mapped to the requested URL-address by DNS. Secondary servers are those that accommodate this object later through our object replicating mechanism. In the above example where a movie is replicated from a server in the US to a server in Europe, the server in the US is the primary server for this movie while the server in Europe is its secondary server. It should be noted here that one server could function as a primary server for a set of objects and as a secondary server for another set of objects at the same time. In order to ensure that new requests can be redirected appropriately, the object-to-server(s) mapping needs to be maintained and exchanged among participating servers whenever hot object replication, or cold object removal or object update happens.

3 Routing of Video Requests

In our scalable video delivery framework, routing of video requestes are done in the following cases.

1. When client selects a movie for the first time: this is the initial case.

2. When the buffer is going to go below the threshold:
   Usually the client does not wait till the buffer is already below the threshold, because we do not want the movie to pause during the process. We name the size of the buffer at which the client selection occurs as *secondary threshold*. The secondary threshold is measured in unit of movie time. Its value equals the size of the (primary) threshold plus the average completion time of a client selection.

3. When the incoming data rate is considerably low:
   If the incoming data rate is much lower than the rate required for normal playback, we immediately launch the server selection process even though the buffer is far from reaching the secondary threshold. This happens when there comes a sudden drop of data rate during the normal playback. However, a server selection does not necessarily mean a switch of servce from one server to another server. If the data rate rises back to the normal level, the client will stop requesting for help.

4. When the user is forwarding or rewinding:
   When the user forwards the movie (or jump to some specific point) there may be a sudden burst of workload on the server because the server may need to process more frames within a short time. Thus,the server may possibly be overloaded. In this situation, we launch the server selection process to identify a possible better server.

One of the following two routing approaches may be used to provide scalable video services transparently:

- Server redirection using the one that is provided by the object-level scalable web framework.
- Client selection

3.1 Server Redirection

Routing of incoming requests are handled in two stages: first using a DNS and then using HTTP redirection, based on servers' workloads and geographical area from which requests are coming in. Since we use HTTP redirection among servers in a Wide Area Network, the redirections can become cyclic in nature. Each HTTP request in this scalable framework is modified to carry a record of its redirection history. Three parameters are needed for redirection history: i) a list of servers to which the request has been redirected; ii) the time of the last redirection; iii) a request ID. We have designed a traceable request redirection approach that examines the redirection history before carrying out HTTP redirections.

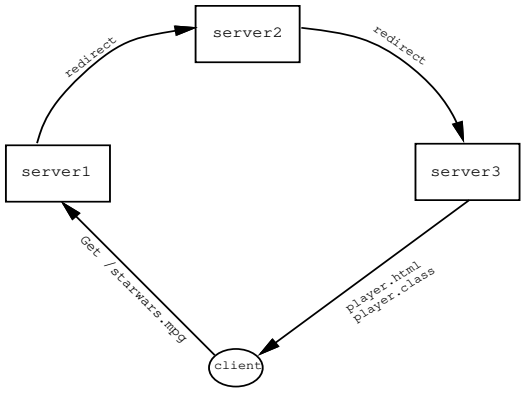

Figure 1: **Server Redirection**

As shown in Figure 1, when a client selects a movie (starwars.mpg) from the list on server1,

a request "GET /starwars.mpg" will be sent to server1 (assuming that server1 is the primary server for the movie starwars.mpg while server2 and server3 are the secondary servers). Then, server1 looks up the object-to-server(s) mapping to find the current hosting servers for this object, aggregates the impact of factors on the overall system response time such as server workload and geographical access pattern, and chooses the best destination server. The server workload such as disk bandwidth is monitored by a local resource monitor on every server and multicasted to other servers periodically. The geographical access pattern is tracked by the access log. Then, server1 redirects the client request with redirection history to server2 (assuming that server2 is the best) by using HTTP redirection. If server2 can admit the new request, then it serves the request by dynamically creating a HTML web page *player.html*, tagged with the media player applet, and sending it back to the client. When the web page player.html arrives at the client's browser, the media player applet will be launched by the browser and starwars.mpg will be retrieved from server2. If the load on server2 is such that the redirected request for the movie cannot be admitted, which may happen due to varying system (network, server) loads, server2 may further redirect the request with redirection history to server3.

Decisions of the algorithm used for selecting a server for an incoming request may not be very accurate due to varying system (network and server) loads. Therefore, multiple redirections may be needed, though they have couple of problems. First, they introduce more overheads than single redirection and the client's response time can suffer. However, when we consider requests for large objects such as video or multimedia lectures, this redirection overhead can be very small compared to the request service time. Second, a request may be redirected among servers in a cyclic fashion. This problem is particularly serious when most of the servers serving the same object are overloaded. Hence, we have added redirection control to coordinate our servers' behaviors so that one request will not be redirected to the same server twice.

The request redirection is done by taking the advantage of code 307 (Temporary Redirect) in HTTP1.1. A server (that cannot handle the incoming request due to high load) inserts the redirection history information into the value field of **Location** in the HTTP response header, which follows the command 307 to indicate the new location of the requested file. Upon receiving response 307, the client will terminate the current connection and proceed to the new location of the requested file. A comparison of server response with redirection history is shown in Table 3.1.

3.2 Client Selection

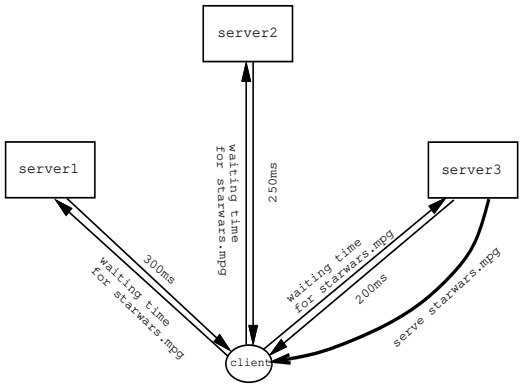

Figure 2: **Client Selection**

In this approach, a downloaded applet can help clients to make server selection. Here, the primary server sends, along with the applet, a list of servers that can potentially handle the client's request. As shown in Figure 2, the applet first sends each participating server a request through client-selection channel to consult the waiting time for playing the movie starwars.mpg. Each available participating server estimates a time according to its current status and replies to the

Server Response	Request to server1	Response from server1	Request to server2
Without redirection history	Get /truelie.mpg HTTP/1.1	HTTP/1.1 307 Temporary Redirect Location: http://server2/mirror/truelie.mpg	Get /mirror/truelie.mpg HTTP/1.1
With redirection history	Get /truelie.mpg HTTP/1.1	HTTP/1.1 307 Temporary Redirect Location http://server2/RH137.132.101.27>>Jan-28-14-02-04>>35/mirror/truelie.mpg	Get /RH137.132.101.27>>Jan-28-14-02-04>>S0135>>/mirror/truelie.mpg HTTP/1.1

Table 1: **Comparison of Server Responses with Redirection History and Without Redirection History**

applet. Also the network delay will be calculated from the round trip time of the request. Then, the applet will retrieve starwars.mpg from the server with the minimum waiting time. The same selection scenario will also happen while handling dynamic user interaction.

Under the default security restrictions, an applet can open network connections only to the host from which the applet itself was downloaded. When the applet tries to open connection to other hosts, it will cause *SecurityException*. We can use one of the following two methods to overcome this problem. The first is to sign the applet with a certificate and request the necessary network access permissions from corresponding servers. The second way is to let the applet talk to a "proxy" program on its host server. That "proxy" represents applet to send and receive data from other hosts. This method protects the server from abuse of privileges by an applet.

The above redirection and selection mechanisms obviously introduce some overheads, but these overheads are usually negligible in comparison with the total request service time since our system is mainly targeted at the websites with large multimedia files.

4 Implementation Experience

The scalable video delivery system is implemented as an object-oriented framework using Java JDK1.2 and JMF 2.0. JMF enables the playback and transmission of RTP streams through a set of APIs. It is developed in a modular fashion, so it can be used to test both client selection and server redirection, without affecting other modules.

4.1 Scalable Servers

The implemented framework consists of multiple Microsoft Windows and Sun Solaris servers. The servers are identified by their network (IP) addresses. A web server in the implemented scalable architecture has the following modules (see Figure 3):

1. **Request Handler:** When a client selects a movie from the list, a HTTP request for that movie file arrives at a server. The request handler makes a decision as to whether the request is to be served or to be assigned to another server. If it serves the request, then it passes the request to the access logger to update the access information. If the served object is a video object, request handler passes the object to video

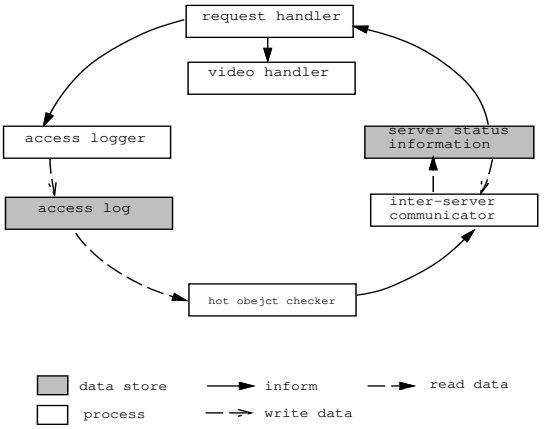

Figure 3: **Functional Modules of a Scalable Server**

handler for processing.

2. **Video Handler:** uses JMF 2.0 for downloading the requested video. It can access a specified video file and deliver the requested frames composing the video.

3. **Access Logger:** updates the access log, collects statistics on HTTP requests, and updates this information in a statistics table, which is the source for a hot object table. The hot object table contains the hot object name and its access information such as bandwidth consumed by each object and the access rate by regions.

4. **Hot Object Checker:** checks hot object table periodically and initiates the procedure of hot object replication and cold object removal.

5. **Inter-server Communication:** module helps the server to communicate with other servers through messages such as requests to replicate/delete/update objects, object-to-server(s) mapping, and the server status information.

4.2 Video Client

When browser users select movies from websites using Internet Explorer (IE), Microsoft Windows Media Player will be launched automatically to display the movies. The Microsoft Windows Media Player is a system default application for IE to support multimedia object processing. This is quite convenient to users, but it cannot easily support on-line presentation control such as fast-forwarding because only a single HTTP connection is setup initially. Furthermore, it cannot intelligently get service from the *best* server if several servers provide the requested movie. Therefore, in order to achieve our goal of scalable video services, we designed a media player applet, which is supported by Java Media Framework (JMF) [9]. JMF is an open media architecture that allows developers to access and manipulate various components of the media playback, synchronization, capture, and transmission. We can quickly build and extend a media player with minimal work. In addition, the use of Java Applet makes the media player platform independent and frees the users from configuring and upgrading the software.

We use Real-Time Transport Protocol(RTP) [11] for transporting real-time data such as audio and video. A RTP session is identified by a network address and a pair of ports. One port is used for media data (RTP data channel) and the other is used for control data (RTCP control channel). RTP is network and transport protocols independent. In our system, we use the UDP for the data channel and TCP for the control channel. TCP is more suitable for the control channel since reliable data transmission is required in the inter-server communication. On the other hand, the overhead of guaranteeing reliable transmission in TCP slows the overall data rate. Therefore, we choose UDP for the data channel in which high data rate is more important.

4.2.1 Client Selection

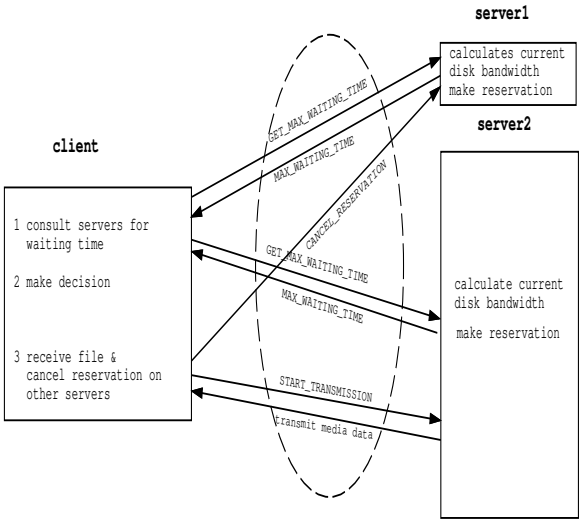

Figure 4: **Serve A Video Request from Multiple Servers**

Besides the RTP session, a client selection session is initiated via TCP connection. It is not practical to do client selection through the control channel of the RTP session although both of them are based on TCP. This is because it is very wasteful to open two channels (one UDP and one TCP) while we are going to use just one (TCP) of them for client selection. When doing client selection(shown in Figure 4, the client firstly checks with each of the available servers about their free bandwidth; secondly, it reserves the bandwidth on one (or some) of the servers; lastly, it launches the file transmission by sending START_TRANSMISSION. Noticing that the TCP connections also consume some system resources on the server, the server may be over flooded by the incoming connections. Therefore, the client must shutdown the connection whenever it is not doing the selection. We designed a protocol for client selection above TCP protocol. The following is a list of the main protocol codes that are implemented at the current stage:

- *GET_MAX_WAITING_TIME:* This code is used by the client to check the maximum available bandwidth on a specific server. Upon receiving this command, the server should reply MAX_WAITING_TIME together with its maximum available bandwidth. A TTL is also send along with the MAX_WAITING_TIME to indicate the valid time interval of the current data. A client can not use the expired data as its reference. During this process, all the participating servers *reserve* the needed disk bandwidth for the client.

- *START_TRANSMISSION:* This command is used to launch the file transmission. In this process, client has selected a particular server to process its request.

- *CANCEL_RESERVATION:* code is used by the client to inform other participating servers that another server has been used by the client for its request. In this process, the other participating servers will release the reserved disk bandwidth. (In the case where servers receive neith *CANCEL_RESERVATION* nor *START_TRANSMISSION*, the servers will release the disk bandwidth after a timeout).

4.2.2 Downloading video from multiple servers

To further improve the playback quality, we allow a client to be served by multiple servers (shown in Figure 5. Each server transmits a specific number of frames (This is done by using the JMF FrameProcessingCotrol.), and the client integrates these frames into one single buffer (This is done by using the JMF MergingDatasource.). The decision on when and how to perform multipoint downloading is made through client selection. If no individual host is capable to serve the movie, the client will send request to multiple hosts (normally two or three depending on

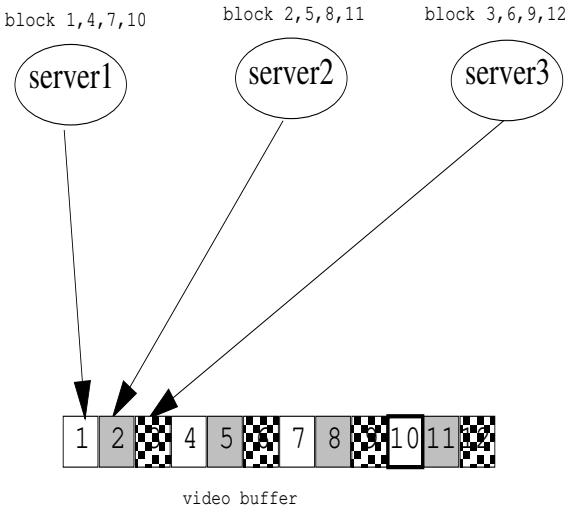

Figure 5: **Client Selection Scenario**

the load condition of the hosts) requesting for certain number of frames. For example, if two servers are going to serve a movie of 10 frames, server1 will send number 0, 2, 4, 6 and 8th frames and server2 will send number 1, 3, 5, 7 and 9th frames. Noticing that some of the frames are dependent on the others (e.g. In MPEG format, B and P frames are dependent on the I frames), the client does not actually request for separate frames. Instead, it requests for the movie blocks each of which is a group of frames between two key frames. Therefore, each movie block can be displayed independently even if the clocks from other servers do not arrive in time.

5 Related Work and Comparison

Existing scalable web architectures use one or more of the following approaches:

- DNS-based selection
- TCP Packet rewriting
- HTTP redirection
- Web page modification (Akamai's FreeFlow)

There are several issues that cannot be effectively addressed by the above approaches:

1. In the first two schemes, full replication of all the content among all the backend servers is needed. This is expensive in terms of space utilization and transfer cost, and every server must have the same capability of processing all kinds of requests coming to any portion of the content the website provides. However, a study about web characterization shows that there is a high concentration of references for the web contents (e.g., 10% of the files accessed on the server typically account for 90% of the server requests and 90% of the bytes transferred [2]).

2. The routing strategies are independent of the service time and resource consumed by the multimedia data such as video and audio. Therefore, it is essential to have a more sophisticated routing mechanism based on the nature of each requested object and the current server workload.

3. FreeFlow approach of Akamai [1] helps draw an object near to the client without full replication, but the website owners need to earmark those objects that will be served by the Akamai network in advance, using a software called Launcher. Therefore, the object set served by the Akamai network is static and the actual bandwidth consumed by this object, which is related to the access pattern and the object size, is not taken into consideration. Hence, if some un-earmarked objects become very popular, the browser viewers might still experience poor response times.

4. Considering request redirection techniques, Network Dispatcher [8] reroutes client requests by rewriting packets. Two-level dis-

patching (DNS plus HTTP redirection) is used in [3, 6], but both of them do not consider the nature of incoming requests. Application-Layer Broker [4] considers the nature of the request and identifies the best resource for the requested object, but the approach does not provide any mechanism for object-level replications. None of these researches declare that they can trace and detect cyclic redirections.

As for providing distributed video service, Tiger system [5] is constructed from a collection of computers connected by a switched network and the data in Tiger is striped across all of the computers and all of the disks. [7, 10] discuss providing distributed video services through building software feedback mechanism. None of them discusses supporting the scalable video services through Internet websites and dynamic object-level replication among servers.

References

[1] "Fast Internet Content Delivery with FreeFlow", Akamai's internal technical report

[2] M.F. Arlitt, C. L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", IEEE/ACM Trans. on Networking, vol.5, no. 5, October 1997

[3] D. Andersen, T. Yang, V. Holmedahl, O.H. Lbarra, "SWEB: Toward a Scalable World Wide Web Server on Multicomputers", Proc. of IPPS'96, Honolulu, April 1996

[4] P. Bai, B. Prabhakaran, A. Srinivasan, "Application-Layer Broker for Scalable Internet Services With Resource Reservation", Proc of ACM Multimedia'99, Orlando, November 1999

[5] W. Bolosky, R. Fitzgerald, J. Douceur, "Distributed Schedule Management in the Tiger Video Fileserver", In Proc. SOSP'09, (www.research.microsoft.com)

[6] V. Cardellini, M. Colajanni, and P.S. Yu, "Redirection Algorithms for Load Sharing in Distributed Web-Server Systems", Proc. 19th IEEE Int'l Conf. Distributed Computing Systems, IEEE Computer Soc. Press, Los Alamitos, Calif., May 1999.

[7] S. Cen et al, "A Distributed Real-Time MPEG Video Audio Player", Proc. of the 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Apr. 1995

[8] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, "Network Dispatcher : A Connection Router for Scalable Internet Services", Proc. of WWW7, Brisbane, April 1998

[9] Java Media Framework API Guide, Nov. 1999

[10] J. Ng et al "A Distributed MPEG Video Player System with Feedback and QoS Control", Proc. of International Conference on Real-Time Computing Systems and Applications, 1998

[11] H. Schulzrinne et al "IETF RFC 1889, RTP: A Transport Protocol for Real Time Applications" http://ww.itef.org/rfc/rfc1889.txt, Jan. 1996