The role of streaming in Interactive Multimedia Documents dissemination

Reetta Pitkänen, Michalis Vazirgiannis, George C. Polyzos

Dept. of Informatics, AUEB
Athens 10434, Greece

reetta@aueb.gr, mvazirg@aueb.gr, polyzos@aueb.gr

ABSTRACT

Multimedia applications require handling of continuous media and support of a variety of media objects with their temporal and spatial relationships. The paper focuses on the importance of streaming usage for the quality of service for multimedia document playout in distributed environments. We describe two Java-based client-server systems for WWW-enabled delivery of Interactive Multimedia Documents (IMDs) supporting a high level of interaction and distribution of scenario and media. We performed series of tests on client-server systems on both Local Area Networks and on Wide Area Networks. The framework has used RTP/RTCP protocols for delivery of continuous streams over the network. The experiments show and quantify the positive effects of streaming on the quality of IMD presentation.

1. Introduction

Multimedia applications require handling of continuous media and support of a variety of media objects with their temporal and spatial relationships. The issue of distributed Interactive Multimedia Documents (IMDs) and specifically their retrieval and execution is an issue of current research [1]. The delivery of interactive multimedia content through the WWW is an upcoming requirement due to the increasing quantity and quality of such content. The rendering of an IMD may become a very complicated task due to the multitude of events occurring and the potentially large set of different presentation options [11].

Multimedia applications require a large amount of resources in terms of host processing power and network bandwidth consumption. Continuous media have an inherent temporal dependency. This results in synchronization requirements for streams, events and groups. It is important for a multimedia presentation system to guarantee the precise presentation time and synchronization between different media objects. Users do not like to wait for a long time for the results of their queries or to see a presentation they requested. Delay in access to multimedia objects used by an IMD can result in significant perceived degradation of the quality of the presentation. Due to high data rates (even compressed video requires high bandwidth) high-speed networks, powerful workstations with real-time facilities and suitable host interface attachments are needed.

Until recently, the standard way to present multimedia documents through a network was to download them first, and then display them. Since transmitting media across a network requires high throughput, start-up latency becomes unacceptable for real-time media. For real-time applications, there is a strong need for streaming media. With streaming media, the client can begin to play the stream without having to wait for the complete stream to download. Some related efforts are the following.

Berkeley Continuous Media Toolkit (CMT) [10] is a framework that consists of a suite of customizable applications that handle streaming media data. In [5] a real-time Multimedia Presentation System (MPS) is presented to experiment with media-on-demand issues using RTP over the Internet with emphasis on video and audio media types. *DirectShow*™ is Microsoft's architecture for capture and presentation of multimedia [6]. DirectShow RTP is a framework that extends the DirectShow architecture, adding support for multimedia applications that stream their data across computer networks using the RTP protocol. The DirectShow RTP framework was designed to support a wide variety of multimedia streaming tasks in a highly extensible manner.

NetMedia is a client-server distributed multimedia presentation system that can flexibly support synchronized streaming of continuous media, in real-time, across the Internet [4]. It is mainly focused on the real-time buffer management and end-to-end data transmission. [3] describes algorithms for effective synchronization between media and the maintenance of a QoS.

In the context of WWW-enabled delivery, special attention should be paid to the Synchronized Multimedia Integration Language (SMIL*)*. The key to HTML success was that attractive hypertext content could be created without requiring a sophisticated authoring tool. SMIL aims at the same objective for synchronized hypermedia [1].

There is an extensive coverage of topics such as streaming video and multimedia presentations. New standard technologies include products such as Real Networks RealSystem Server [14] and the MS Windows Media Server [13]. However, the merging of these two areas, combined with substantial user interaction, has received little attention by researchers. Also, most of the research has concentrated on streaming of a single media object.

This paper describes the architecture of two Java-based client server systems for WWW-enabled delivery of IMDs supporting a high level of interaction and distribution of scenario and media. We performed series of tests on client-server systems on both Local Area Networks (LANs) and on Wide Area Networks (WANs). The system has been implemented in Java using the Remote Method Invocation (RMI) [9] client-server communication protocol, Java Media Framework (JMF) [2] for handling multimedia objects and RTP/RTCP protocols for delivery of continuous streams over the network.

2. System architecture and implementation

We have developed two approaches for IMD delivery and rendering: without streaming (the more traditional one) and with streaming. Rendering is the process of the actual presentation of the media, i.e. where, when, for how long and under what transformations each media object will be presented [11]. The first system (Figure 1) retrieves scenarios from an IMD server and media objects from a set of http servers.

An IMD involves a variety of individual multimedia objects presented according to a set of specifications called the IMD scenario [11]. A scenario is a set of autonomous functional units called scenario tuples, which describe the flow of the application in spatial and temporal domains (e.g. “video1> 4 image1> 0 button1> 5 video 1||” which means “start video1, after 4 sec start (show) image1, immediately (after 0 sec) start button1, after 5 sec pause video1”). It also describes how application and system events are handled. The scenario defines all media objects used in a multimedia presentation. It does not include the media object data, but only references to their location in the form of URLs [2].

During the authoring phase, the author of an IMD defines the spatial and temporal order of the media objects within the document context and the relationships between these objects. The author also defines how an end user will interact with the presentation as well as the way application or system events will be treated. During an IMD session, whenever a media object is to be presented, the client communicates with the respective http server and presents the media directly from the remote machine without storing it locally. It is important to notice that the client will present the media only when it has retrieved the entire media object. A detailed description of this system can be found in [1].

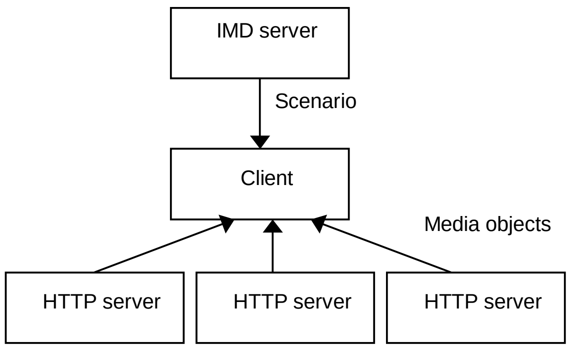

Figure 1. Architecture for client-server system without streaming

In both approaches, the system implementation is based on Java; the IMD server, the Media server and the client are all implemented in Java, and therefore portable across platforms. Java has many appealing features such as built-in multi-thread support and cross-platform compatibility. Moreover, all WWW-browsers support Java. Therefore, it is possible to present an IMD in any WWW-browser. Both implementation frameworks support concurrent processes (i.e., a set of instruction streams may run in parallel).

In the approach without streaming, the client retrieves the continuous media (video and sound) from http servers and presents them with the aid of the JMF. The JMF specifies a unified architecture, messaging protocol and programming interface for media players, media capture and conferencing [1]. JMF APIs support the synchronization, control, processing and presentation of time-based media.

The second system is based on an approach that uses streaming of continuous media (Figure 2). It is an extension of the system described in [1]. With streaming media, the client can begin to play the stream without having to wait for the complete stream to download. The streaming is implemented using the Real-time Transport Protocol (RTP). RTP is a protocol providing support for applications with real-time properties, including timing reconstruction, loss detection, security and content identification. RTP enables the identification of the data being transmitted, determines the order the packets should be presented in and synchronizes media streams from different sources [12]. JMF APIs enable the transmission and playback of RTP streams.

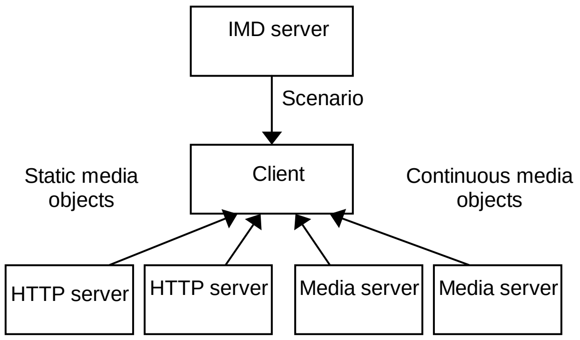

Figure 2. Architecture for client-server system that uses streaming of continuous media

The following modules provide the server functionality. The IMD server is responsible for the delivery of IMD objects (scenarios) to the client. The RMI registry, which is the naming service of RMI, is used to establish communication between client and server [1]. The Media server (Figure 3) is responsible for the delivery of continuous media streams (video and audio) to the client. Finally, a set of http servers store and serve the hypermedia objects.

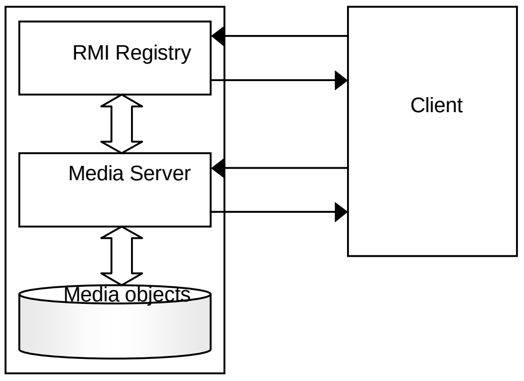

Figure 3. The media server architecture and the communication with the client.

When the media server is started, it registers itself to the RMI Registry and waits for client requests. When there is a client request, it uses a Processor to produce an RTP-encoded DataSource and to construct a SessionManager. The SessionManager is used to coordinate an RTP session as well as to keep track of the session participants and the streams that are being transmitted. It also handles RTCP control channels [2]. The system supports simultaneous media streams.

Each media type is transmitted in a separate RTP session. If a video clip contains both visual component and audio component, they are transmitted as separate RTP streams. The media data for a session is transmitted as a series of packets called the RTP stream. The RTP data packets are not guaranteed to arrive in order. In fact, they are not guaranteed to arrive at all [12]. The receiver is responsible for the reconstruction of the packet sequence and the detection of lost packets.

RTCP is a control protocol that works in conjunction with RTP [12]. The primary function of RTCP is to provide information to an application regarding the quality of data distribution. Each RTCP packet contains sender and/or receiver reports that report statistics useful to the application. These statistics include number of packets sent, number of packets lost, inter-arrival jitter, etc. This reception quality feedback will be useful for the sender, receivers, and third-party monitors [2]. For example, the sender may modify its transmission rate based on the feedback; receivers can determine whether problems are local, regional or global; network managers may use information in the RTCP packets to evaluate the performance of their networks for multicast distribution. The sender report (SR) contains the total number of packets and bytes sent as well as information that can be used to synchronize media streams from different sessions. A receiver report (RR) contains information about the number of packets lost, the highest sequence number received, and a timestamp that can be used to estimate the round-trip delay between the sender and the receiver.

Users have control over the buffer maintained by the RTP receiver. The buffer has two parameters that can be controlled: Length and Minimum threshold. The Length is the length of the buffer. When the data in the buffer exceeds this length, the data at the head of the queue (earlier data) is dropped. The Minimum threshold determines a point in the buffer. Data is forwarded to the application only when enough data comes to the buffer so as to reach the min. threshold point. According to [2] RTP video buffer limit in frames is 4 frames, while audio buffer limit is a maximum of 1000 ms of audio. The communication between the client and the servers is performed exclusively (except for the actual streaming of media) using the Remote Method Invocation (RMI) protocol [1].

3. Experimental evaluation

We carried out extensive experiments in LAN configuration consisting of three machines connected by a 10 Mbps Ethernet. We used one media server and two clients. The media server is on the same machine as the http server that holds the media objects. It is important to notice that the scenarios are time-independent and usually a few kilobytes in size, so it not very important where the IMD server is located (can be either on the same machine with the client or on a remote server). Table 1 summarizes the different client configurations we used.

	Media server (Windows 2000)	WAN Client (Windows 98)	LAN Client 1 (Windows 98)	LAN Client 2 (Windows 2000)
CPU	Pentium III 533 MHz	Pentium II 400 MHz	Pentium 200 MHz	Pentium III 650 MHz
Memory	256 MB	128 MB	48 MB	128 MB

We used several different scenarios for the experiments. These scenarios contain different combinations of video, audio, images and text. We used 240 x 180-pixel resolution video clips and audio with 22.5 kHz sampling rate. We also used gif and jpg files for images and txt files for text. Each scenario was executed four times concurrently on each client, using three different execution schemas (no streaming, streaming with maximum values for buffer and minimum threshold and streaming with default values for buffer and minimum threshold). Due to space limitations we will refer to selected test results that are representative of the overall system behavior.

Two parameters were taken into consideration for the experiments: end-to-end delay for each actor and packet loss. In this case, delay is defined, as the amount of time needed from the moment a client requests a media object until the moment its presentation is started. In order words, delay is a sum of server latency, network latency and client startup latency. More specifically, the different delays involved in end-to-end delay estimation are: (i) data retrieval delay, attributed to disk access delays, (ii) packetization delay, (iii) network delay, including propagation and queuing delays, (iv) depacketization delay, and (v) decoding and rendering delays. Results for streaming is measured with two different values for buffer length and minimum threshold in order to verify how much these values affect the parameters. In the first case (maximum values), the buffer length is set to 270 ms and 135 ms for minimum threshold for video while sound has the corresponding values of 1000 and 500. In the second case (default values), the buffer length is set to 135 ms and 0 ms for minimum threshold for video while sound has the corresponding values of 250 ms and 125 ms. We tested four different scenarios on local network using two clients, and other two scenarios were tested over the internet using a remote client.

3.1. Test results

In this section we describe the experimental results using an interactive scenario: “Exploring the heavens” which contains a series of images, text and video clips. The action list of scene1 is "ALKUR_VIDEO> 0 NEXT_BTN1>". Action list of scene2 is "EARTH_IMG> 4 EARTH_IMG< 1 MOON2_IMG> 4 MOON2_IMG< 1 IO_IMG> 4 IO_IMG< 2 SPACE_VIDEO> 1 MISSION1> 2 NEXT_BTN2> ". Finally, the action list of scene3 is "A17> 3 A16> 5 DISCOVERY> 3 EXITBTN>". The temporal ordering of the above media objects and scenes appear in Figure 4.

No streaming: The first time the scenario is executed, the delay for video clip ALKUR_VIDEO is much bigger than with the next three executions, while the video clip SPACE_VIDEO has approximately the same delay for all four executions. Packet loss is small and is not noticeable by the user.

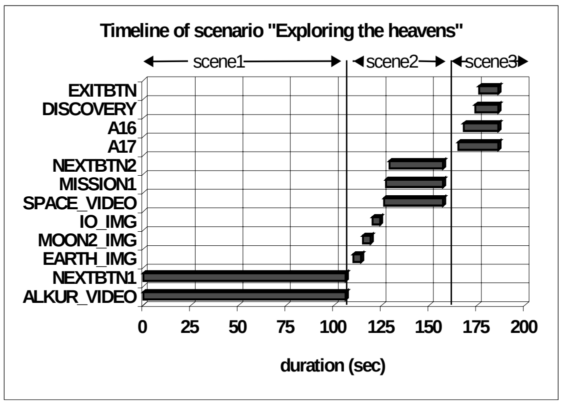

Figure 4: Timeline of the scenario “Exploring the heavens”

Streaming (max buffer length and threshold): Delay is much smaller than with no streaming. The first time the scenario is executed, the delay for video clip ALKUR_VIDEO is much bigger than with the next three executions, while the video clip

SPACE_VIDEO has approximately the same delay for all four executions. Packet loss is not noticeable by the user and is smaller than with no streaming.

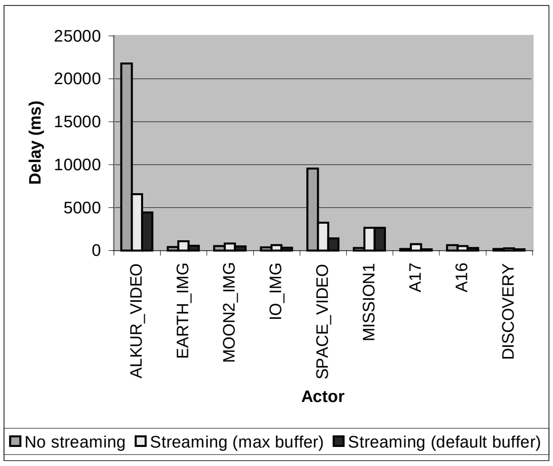

Figure 5: Average delays for actors of scenario "Exploring the heavens" environment LAN, client 1, average delay for 4 executions

Streaming (default buffer length and threshold): Delay is much smaller than with previous two schemes. The first time the scenario is executed, the delay for video clip ALKUR_VIDEO is much bigger than with the next three executions, while the video clip SPACE_VIDEO has approximately the same delay for all four executions. Packet loss is not noticeable by the user and is smaller than with no streaming and approximately equal than with streaming with maximum values.

It is important to notice that text "MISSION1" that appears one second after video "SPACE_VIDEO" has very large delay when streaming is used. Playback quality is good and is approximately the same with all execution schemes.

4. Conclusions and future work

We have presented two Java-based client-server systems for IMDs that support a high level interactivity and distribution of scenario and media. Using RTP/RTCP we have developed our system to support streaming of continuous media objects. The major conclusions are the following:

As was shown by our experiments, the processing power of the client can become a bottleneck and influence the performance of the presentation a great deal. The processing power affects both client start-up latency and playback quality of the presentation.

The approach that uses streaming has significantly smaller delay than the approach with no streaming. Additionally, the client buffer length and minimum threshold affect the client start-up latency.

Streaming improves the synchronization of the media objects. Best performance in terms of delay and synchronization was achieved using streaming with default values for buffer and threshold (jitter buffer).
It was shown than when no streaming is used, the network becomes a bottleneck. On the contrary, when streaming is used, client-processing power becomes the bottleneck. Finally, streaming also decreases the percentage of lost packets.

We extended our experiments in a WAN configuration consisting of a remote client and media server and the results were inline with the above-mentioned results.

We plan to extend the client-system architecture to support QoS features. We also plan to include other features such as pre-fetching into our system. The idea of pre-fecthing is to predict data access needs in advance so that a specific piece of data is loaded before it is actually needed by the application. The need for pre-fetching in our system is based on the observation that there are frequent periods of time during which the bandwidth is under utilized or not utilized at all. In order to achieve high quality IMD presentations over the Internet, protocols such as RSVP and Differentiated Services (DiffServ) should be considered.

References

[1] Th. Markousis, D. Tsirikos, M. Vazirgiannis, Y. Stavrakas: "WWW-enabled delivery of interactive multimedia documents," *Computer Communications* 23 (2000) 242-252.

[2] Sun Microsystems, Inc: "Java^{TM} Media Framework API Guide", http://java.sun.com/products/java-media/jmf/2.1/guide/, November 19, 1999.

[3] T. V. Johnson, A. Zhang: "A Framework for Supporting Quality-Based Presentation of Continuous Multimedia Streams", Proceedings of the 1997 International Conference on Multimedia Computing and Systems (ICMCS '97), Ottawa, Canada, June 1997.

[4] Y. Song, M. Mielke, A. Zhang: "NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments", Proceedings of the ACM Multimedia, Italy, Florence, June 1999.

[5] S. Palacharla, A. Karmouch, S. A. Mahmoud: "Design and Implementation of a Real-time Multimedia Presentation System using RTP", Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference, Washington, DC, August 1997

[6] L. S. Cline, J. Du, B. Keany, K. Lakshman, Christian Maciocco, David M. Putzolu: "DirectShow RTP Support for Adaptivity in Networked Multimedia Applications", IEEE Multimedia Systems '98

[7] F. Rousseau, A. Duda: "Streaming Support in an Advanced Multimedia Infrastructure for the WWW", Proceedings of the Fourth IEEE Symposium on Computers and Communications

[8] J. Du, M. Clark, D. Putzolu, D. Ryan, L. Cline, D. Newell: "An Extensible Framework for RTP-based Multimedia Applications", In Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video (St. Louis, MO, May 1997), IEEE, pp. 53--60

[9] Sun Microsystems, Inc: "Java^{TM} Remote Method Invocation", http://java.sun.com/products.jdk/rmi/

[10] M. H. Jackson, J. E. Baldeschwieler, and L. A. Rowe: "Berkeley Continuous Media toolkit API", September 1996.

[11] M. Vazirgiannis, "Interactive Multimedia Documents", Springer-Verlag, LNCS Series, October 1999, ISBN 3-540-66711-3.

[12] RFC 1889 "RTP: A Transport Protocol for Real-Time Applications" http://www.faqs.org/rfcs/rfc1889.html

[13] Microsoft: Windows Media Server: http://www.microsoft.com/windows/windowsmedia/EN/default.asp

[14] Real Networks: RealSystem Server: http://www.realnetworks.com/products/basicserver/info.html