NetMedia: A Client-Server Distributed Multimedia Database Environment

Sreenivas Gollapudi and Aidong Zhang
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
{golla-s, azhang}@cs.buffalo.edu

Abstract

Advances in multimedia computing technologies offer new approaches to support on-line accesses to information from a variety of sources such as video clips, audio, images, and books. A client-server distributed multimedia system would be a practical approach to support such functionalities. In this paper, we present the design and implementation of a client-server distributed multimedia database environment that can be used to support large digital libraries. System architecture and design are described. Server functionalities, including client scheduling, data buffering and admission control, are investigated. A client request can only be admitted if both the quality-of-service (QoS) requirements from the client and the upper bound on total buffer consumption at the server are maintained.

1 Introduction

Increased networking capabilities have made it easier and less expensive to tie together different types of computers and personal workstations. In addition, current computer technology supports many multimedia standards for software and hardware to uniformly handle multimedia data. These technologies have made it possible to establish a distributed multimedia database environment which supports on-line accesses to various multimedia data resources. A distributed multimedia database system integrates multiple local media database sites into a single global unit. Each site has its own media database management system, connected via one or more LANs (Local Area Network) and/or a WAN (Wide Area Network). In such a system, a user at any network node can log on and access data anywhere in the global system as though all data resided right at the user's local site.

New issues appear in such distributed multimedia environments. Many multimedia applications, such as recording and playback of motion video and audio, slide presentations, and video conferencing, require continuous presentation of a media data stream and the synchronized display of multiple media data streams.

Because of these time-related requirements, the allocation of various system resources becomes the most important aspect of the system design. Specifically, a successful system must maintain balanced tradeoffs between granting client requests and the overall performance of the system. On the one hand, the participating clients make their requests and expect these requests to be granted by the server. On the other hand, the server wants to maintain balance among all clients and maximally utilizes all system resources.

New system architecture and protocols must be designed to achieve the above goals. Some similar functions may need be implemented at both client and server sides to achieve the best system performance. At the client side, the synchronized presentation of multiple media streams must be maintained. A client may have to make decisions on skipping or dropping some data to maintain the synchronization of presentations. Buffer management at the local side may be needed to ensure hiccup-free presentations in case of network delays. In addition, clients are responsible to send data delivery requests along with the quality-of-service (QoS) requirements. A client may also periodically send data delivery rate information to the server to vary the rate at which data are sent if a change is needed in the presentation schedule. At the server side, protocols to address scheduling of multiple client requests, buffer management for data allocation and replacement, and admission control of clients must be carefully designed.

In this paper, we will focus our attention on the design of the entire system and the server functionalities. The design and implementation of a client-server distributed multimedia database environment that can be used to support large digital libraries will be presented. We will describe the details on the system architecture and design. Protocols to achieve server functionalities, including client scheduling, data buffering and admission control, are investigated. A client request can only be admitted if both the quality-of-service (QoS) requirements from the client and the upper bound on total buffer consumption at the server are reserved.

The remainder of this paper is organized as follows. Section 2 introduces the system architecture. In Section 3, we present the system design strategies, including the data model, QoS parameter specification and buffer model. In Sections 4, we investigate the approaches for the server to schedule client request, maintain upper bound of buffer consumption and control the number of client requests. Concluding remarks are offered in Section 5.

2 System Architecture

The system architecture under consideration is illustrated in Figure 1. This architecture includes a distributed multimedia server, a set of multimedia databases, and a set of clients which access the server. The multimedia system server is distributedly superimposed on top of a set of database management systems (DBMSs) and file systems. As certain media streams may be represented and stored in different formats, the underlying DBMSs or file systems can be heterogeneous. The main function of each client at a workstation is to display multiple media data to the user in the specified format. Such an architecture can provide adequate database support for multimedia applications demanding script-based interactive multimedia presentations [TK95].

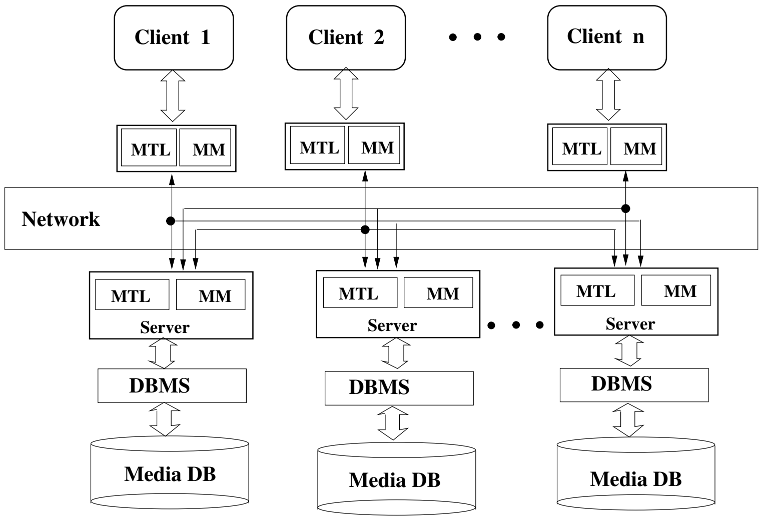

Figure 1: NetMedia system architecture

As shown in Figure 1, the distributed multimedia database management system contains two main modules: a multimedia task language (MTL) interpreter and a media manager (MM). At each client site, the multimedia task language MTL interpreter allows users to specify a set of tasks associated with a multimedia task, including synchronization requirements on component tasks. A multimedia task specified in MTL is then processed by the interpreter, and data accesses are sent to both the MM and the underlying DBMS or file system for processing.

The MM component at each server site supports the multi-user aspect of media data caching and scheduling. It maintains real-time retrieval of media data from the multimedia database and transfer the data to the client sites through network. The MM at a client site ensures that media data stored in the multimedia database will be available on demand in the local buffer and the synchronous presentation of multiple media streams.

The detailed design of these system components are offered in the following subsections.

2.1 Multimedia Server

The client-server architecture consists of multimedia servers and multimedia clients. The multimedia server, henceforth referred as the server, sits on top of a file or database system. The server is an event driven process that uses a time-ordered prioritized query queue to service requests. Figure 2 shows the organization of the server.

The server is client-driven [LD93], that is, all data transmitted from the server to the client is explicitly requested. It is the job of the server to guarantee sufficient bandwidth. The clients have to read data at the same rate as the transfer rate to minimize the buffer requirements at the client. The interpretation of the media data falls on the clients as data is presented at the client workstations. At present, three types of data, images,

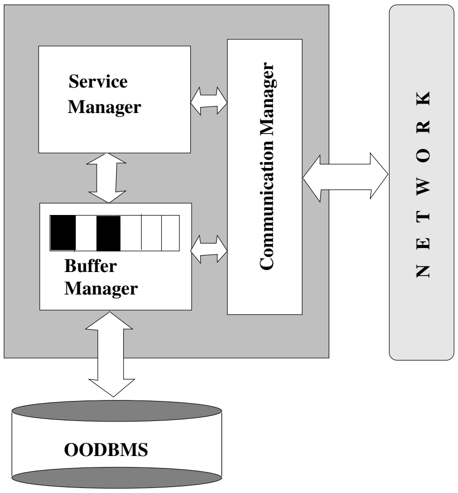

Figure 2: Server configuration

digital video and audio are supported.

The server has three components, namely, a *communication manager*, a *buffer manager*, and a *service manager*. The receiving of queries and sending of data to the clients is handled by the communication manager. It is the responsibility of the communication manager to transfer data at the rate requested by each client. This ensures that all data is scheduled to meet the deadline requirements of clients. The communication manager is multi-threaded. It starts a different service thread for every concurrent stream being serviced. The service manager handles the scheduling of service threads. The retrieval of media data and storage of data in the buffer is the responsibility of the buffer manager. The handling of multiple client requests and buffer management are detailed in Section 4.

The server and the clients exchange information via a virtual circuit connection (TCP). At present the server and the clients sit on an LAN (Local Area Network). Clients control the rate at which the server must transmit data to the clients. Clients set the rate at the beginning of each connection and can change it at the beginning of a new request.

2.2 Multimedia Client

A client is responsible for synchronizing the images, audio and video packets and delivering them to the output devices on the client workstation. A client makes decisions to skip or drop frames to maintain the synchronization of presentations. In addition, a client sends rate information to the server to vary the rate at which data are sent if a change is needed in the presentation schedule.

Each client has four components: a *service manager*, a *scheduler*, a *buffer manager*, and a *communication manager*. The communication manager receives the data from the server, unpacks it and sends it to the buffer manager. The communication manager also sends any rate information it receives from the scheduler to the server. The service manager deals with the reading of data from the buffer and delivering it to the appropriate output device. This service is multi-threaded. The service manager starts a different thread for every media stream that is to be presented. The scheduler is responsible for synchronizing the delivery of media objects in all the participating media streams. The buffer manager, as before, deals with the management of data in the buffer. A buffer model that is the basis for buffer management at the client and server sites is presented in the next section. Figure 3 shows the logical layout of clients. Both scheduling and buffer management at the client site will not be discussed in this paper [GZ96, ZG95a].

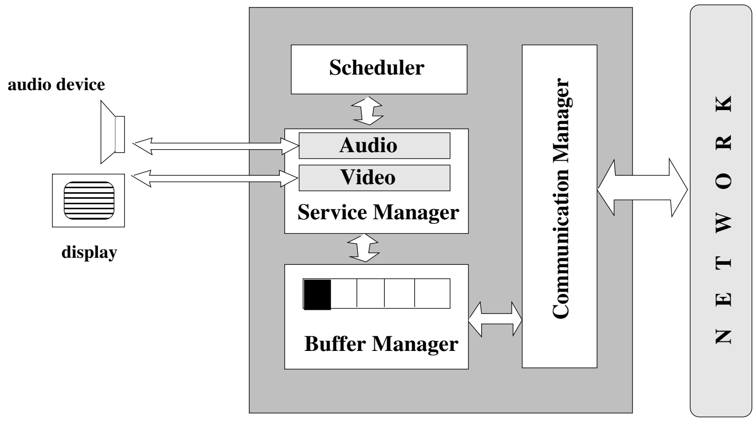

Figure 3: Client configuration

3 System Design

In this section, we will introduce the system and data models that will be used in the rest of the paper. Quality of service (QoS) parameters and a buffer model will be introduced.

3.1 Data Model

A media stream can be viewed abstractly at several levels. At the lowest level, a media stream is viewed as an unstructured *BLOB* (binary large objects) into several higher-level object classes. Objects from different media streams may also be spatio-temporally combined into multimedia objects. Several conceptual data models which follow this general scheme have been proposed. However, few efforts have been made to formalize a multimedia data model at the task management level for the purpose of scheduling media data operations.

In the proposed data model, we assume that each media stream is broken into a set of atomic objects. Higher levels of object classification need not to be considered in this context. Each atomic object represents a minimum chunk of the media stream that bears some semantic meaning. Atomic objects in different media streams may have different internal structures. For example, a continuous video stream can be segmented into a set of atomic objects, each of which contains a set of video frames with specific semantic meaning. Similarly, a continuous audio stream can be segmented into a set of atomic objects, each of which contains a set of audio samples with specific semantic meaning.

The atomic objects within a media stream are linked together through intra-synchronization time constraints. These constraints may specify discrete, continuous, overlapping, or step-wise constant time flow relationships among the atomic objects. For example, some multimedia streams, such as audio and video, are continuous in nature, in that they flow across time; other data streams, such as slide presentations and animation, have discrete, overlapping, or step-wise time constraints. It may, for example, be necessary to display two distinct slide objects jointly within a single slide presentation stream. In general, the temporal relationship between two atomic objects in a single stream may conform to any of the thirteen temporal relationships described in [All83]. In our representation, each atomic object is associated with a relative start time and a time interval which specifies the duration of its retrieval, with the initial atomic objects in the media stream assumed to start at time zero. The actual start time of a media object is usually dynamically determined. Once a media stream is invoked, it is associated with an actual *start time*; each media object within that stream will similarly be associated with an actual start time. We use $< o, t, \triangle t >$ to denote that object o is to be delivered at time t and will last time period $\triangle t$.

3.2 QoS Specification

Little and Ghafoor [LG90] have proposed several parameters to measure the QoS for multimedia data presentation. The following parameters have been listed: (1) average delay, (2) speed ratio, (3) utilization, (4) jitter, and (5) skew. The *average delay* is the average presentation delay of each object in a time interval. The *speed ratio* is the actual presentation rate to the nominal presentation rate. The *utilization* equals the ratio of the actual presentation rate to the available delivery rate of a set of objects. Ideally, both the speed and utilization ratios should equal 1. During the presentation of a video stream, frame duplication leads to utilization values greater than 1, while dropping frames would lead to values less than 1. The *jitter* is the instantaneous difference between two synchronized streams. The *skew* is the average difference in presentation times between two synchronized objects over n synchronization points. Clearly, average delay, speed ratio, and utilization are used to measure the quality of individual media stream presentations, whereas jitter and skew are used to measure the quality of presentation among multiple media streams.

While the delivery of each media stream would ideally minimize the average delay and hold the parameters of speed ratio and utilization close to 1, the achievement of these three goals is actually in conflict. There must therefore be trade-offs between these goals during scheduling. Consider a synchronous presentation of audio and video streams. If the scheduler attempts to minimize the average delay of audio objects, it must respond to the delay of an audio object by dropping some frames in the corresponding video object. If the scheduler tries to hold the utilization of video objects close to 1 when delays occur, it must decrease the speed ratio of these objects, and, consequently, increase the average delay. Thus, it is generally impossible for all parameters to achieve an ideal state for all applications. There must be trade-offs among different QoS parameters.

Different application domains may have different QoS requirements. The specific QoS requirements for the domain of education or training in educational digital libraries were discussed in [ZG95b]. We will not discuss here the QoS specification for specific application domains. We assume that maximum allowable delays for individual media streams are pre-specified. These measures provide the permissible ranges for average delay and speed ratio. In addition, we assume that maximum allowable skips for individual media streams are also pre-specified. These measures provide the permissible ranges for utilization.

3.3 Buffer Model

We now present a framework which generates the required start times for media objects. This framework guarantees the continuity of media stream presentation while minimizing buffer utilization at both client and server sites.

3.3.1 Media Streams

Buffer management is needed in both client and server sites to ensure that the loading of media objects will not cause the delay of their presentation. At the client sites, to facilitate a hiccup-free presentation, we must ensure that an object is present in local memory before it is delivered. At the server sites, we must ensure that once a media stream is started to be retrieved and transmitted, this retrieval and transmission will be performed in a desirable rate.

At the server site, the loading of a media stream from disk to memory is much faster than the transmission of the media stream across the network to the client site. However, at the client site, the rate at which data is received from the servers could be comparable to the display rate of the media streams. Therefore, both network and storage delays must be considered in determining the preloading time of a media stream at the client sites.

We will now define a general model that can be used at both the client and server sites. For the sake of brevity, we will use *delivery/consumption* interchangeably to refer to delivery of a media stream at the client site as well as the transmission of a media stream from the server site. Similarly, *loading* will be used to refer to loading of media data from the disk into the buffer at the server site as well as loading of media data into the buffer at the client site. Delays at the client site include network delays during transmission and the storage delays at the servers. Multiple streams at the client site refers to the streams that are involved in a presentation at the client site, while multiple streams at a server refer to the multiple requests a server can service simultaneously.

Let $t_{l_s}^m$ be the time at which the loading of media stream m begins and *loading function* $L_m(t, t_{l_s}^m)$ be the total number of media objects of m read at time t. Let $t_{c_s}^m$ be the time at which the consumption of data stream

m begins and *consuming function* $C_m(t, t_{c_s}^m)$ be the total number of media objects consumed at time t. The number of media objects that must be buffered at any given time is then given by

$$B_m(t, t_{c_s}^m) = \begin{cases} 0 & \text{if } t < t_{l_s}^m, \\ L_m(t, t_{l_s}^m) & \text{if } t_{l_s}^m \le t < t_{c_s}^m, \\ L_m(t, t_{l_s}^m) - C_m(t, t_{c_s}^m) & \text{if } t \ge t_{c_s}^m. \end{cases} \tag{1}$$

Given D_{max} as the amount of buffer for delay recovery in the delivery of media stream m, this amount must be added to consumption to determine the start time of delivering the stream.

Suppose that a solution is to begin delivery at time x. That is, $B_m(t,x)$ is at least zero for any time $x \le t \le t_{l_f}^m$, where $t_{l_f}^m$ is the time at which the loading of data stream m is completed. If we compare $B_m(t,x)$ with $B_m(t, t_{l_s}^m)$ in the range $x \le t \le t_{l_f}^m$, we see that

$$\begin{aligned} B_m(t,x) &= L_m(t, t_{l_s}^m) - (C_m(t,x) + D_{max}) \\ &= L_m(t, t_{l_s}^m) - C_m(t,x) - D_{max} \\ &= B_m(t, t_{l_s}^m) + C_m(t, t_{l_s}^m) - C_m(t,x) - D_{max} \\ &= B_m(t, t_{l_s}^m) - D_{max} + C_m(t, t_{l_s}^m) - C_m(t,x). \end{aligned} \tag{2}$$

If $B_m(t, t_{l_s}^m) \ge D_{max}$ in the range $t_{l_s}^m \le t \le t_{l_f}^m$, then $B_m(t,x) \ge 0$ for $x \ge t_{l_s}^m$. Thus, the start time of stream m can be $t_{l_s}^m$. We now consider the situation that $B_m(t, t_{l_s}^m) - D_{max}$ may be negative in the range $t_{l_s}^m \le t \le t_{l_f}^m$. Let $B_m(t, t_{l_s}^m) - D_{max} \ge -k(k \ge 0)$ and $B_m(t,x) \ge 0$. Thus, x must be the minimum start time such that

$$C_m(t, t_{l_s}^m) - C_m(t,x) \ge k. \tag{3}$$

Thus, x can be determined when both consuming function and the loading time are given. This start time assumes that the entire stream will be continuously loaded. However, in our context, we assume that the data unit to be accessed is media object rather than the entire stream. Thus, after the display or deliver time of stream m is determined, the deliver time of each media object within the stream must also be precisely determined.

3.3.2 Media Objects

In order to ensure a hiccup-free presentation based on pre-determined time, the loading of each media object must guarantee that there is enough object data to be consumed at its consuming time. The presentation of each object can thus be divided into two phases: (a) a loading phase and (b) a consumption phase. Let $t_{l_s}^{im}$ be the time at which the loading of object o_{im} of stream m begins and *loading function* $L_{im}(t, t_{l_s}^{im})$ be the amount of object o_{im} of stream m read at time t. Let $t_{c_s}^{im}$ be the time at which the consumption of object o_{im} of stream

m begins and *consuming function* $C_{im}(t, t_{c_s}^{im})$ be the amount of object o_{im} consumed at time t. The amount of buffer space that must be allocated for object o_{im} at any given time t is given by

$$B_{im}(t, t_{c_s}^{im}) = \begin{cases} 0 & \text{if } t < t_{l_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) & \text{if } t_{l_s}^{im} \le t < t_{c_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) & \text{if } t \ge t_{c_s}^{im}. \end{cases} \tag{4}$$

In similar manner to the derivation given above, we can derive the relationships between the loading and consuming times for each media object o_{im}, as follows:

$$\begin{aligned} B_{im}(t, t_{c_s}^{im}) &= L_{im}(t, t_{l_s}^{im}) - (C_{im}(t, t_{c_s}^{im}) + D_{max}^{im}) \\ &= L_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) - D_{max}^{im} \\ &= B_{im}(t, t_{l_s}^{im}) + C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) - D_{max}^{im} \\ &= B_{im}(t, t_{l_s}^{im}) - D_{max}^{im} + C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}). \end{aligned} \tag{5}$$

Let k_{im} be determined by $B_{im}(t, t_{l_s}^{im}) - D_{max}^{im} \ge -k_{im}(k_{im} \ge 0)$, where D_{max}^{im} is the amount of buffer for delay recovery in the presentation of the object o_{im}. We then have

$$C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) \ge k_{im}. \tag{6}$$

Thus, for each object o_{im} in stream m to be successfully presented at time $x = t_{c_s}^{im}$, it must be loaded into memory at a time satisfying Formula (6).

In case that the consuming function is linear, that is,

$$C_m(t, x) = \begin{cases} 0 & \text{if } t < x, \\ r_c^m(t - x) & \text{if } t \ge x. \end{cases} \tag{7}$$

where r_c^m is the consuming rate of stream m. We then have the start time of the first object of the stream m based on Formula (3):

$$x \quad \ge \quad t_{l_s}^m + \frac{k}{r_c^m}, \tag{8}$$

and the preloading times for the rest of the media objects based on Formula (6):

$$t_{l_s}^{im} \le t_{c_s}^{im} - \frac{k_{im}}{r_c^m}, \tag{9}$$

Depending on the loading delays and the amount of data that has to be loaded, $t_{l_s}^{im}$ can belong to any of the time periods in which previous media objects are consumed.

4 Multimedia Server Functionalities

In this section, we will investigate protocols for the scheduling of client requests, buffer management for data allocation and replacement, and admission control of clients.

The distributed multimedia server plays a central role in the delivery of continuous media in distributed systems. The server handle the retrieval, storage and delivery of media data to clients across the network. Playback of digital video and audio requires high data delivery rate. Continuous playback of such media data necessitates hiccup-free presentation and is usually coupled with the delivery of another media stream via synchronization. Thus, the delivery of continuous media can be thought of as a sequence of periodic tasks with deadlines.

Since the fetching rate of continuous media is much higher than the delivery rate, it is conceivable to have systems that fetch data just before playback. However, owing to the bursty nature of retrieval, information retrieved may have to be buffered before playback. It is the function of the server to store enough data in the buffers so as to allow for a continuous playback at the client process. Having a modest amount of buffering would be enough to meet the deadline requirements for single client requests. In practice, however, a distributed multimedia server has to process more than one client request simultaneously. This puts more constraints on shared resources like the server cache. Guaranteeing the deadline constraints of a client can be achieved by simply dedicating a connection to the disk per client. However, this can seriously limit the number of client requests at any time owing to limited number of disk access channels available. More client requests can be scheduled by multiplexing disk access across multiple client requests. This requires efficient buffer management and admission control by the server to maximize the number of client requests serviced. In fact, it can be shown that the maximum number of client requests can be serviced simultaneously by the server when the amount of data retrieved for each client is proportional to the consumption rate of the client [GC92, LD93, HGP94].

4.1 Scheduling Multiple Clients

Consider a distributed multimedia server servicing n client requests simultaneously (say $R_1, R_2, \ldots R_n$, respectively). At the outset, let us assume that a request R_i identifies a media object Obj_i that a client wishes to present. Furthermore, this object could be stored as a number of segments in a local DBMS. The segments are assumed to be of the same size. It is the responsibility of the server to identify all the relevant segments and load them into the buffer before transmitting them to the client.

The service time of the server is divided into fixed size of intervals. Each interval is then dynamically divided into n rounds, assuming that there are currently n client requests. The server services all current n requests in terms of periodic *rounds* [HGP94], retrieving a fraction of the segments for each client request in each round. Thus, interval I_l consists of $round_{lj}$, $j = 1, \ldots, n$. The number of segments to be retrieved in each round can be calculated as follows. Let $r_{c_1}, r_{c_2}, \ldots, r_{c_n}$ be the playback or consumption rates[1] specified by the n client requests, respectively. Without loss of generality, let the duration of each interval be t. The duration of each round in an interval with n client requests would then be

$$d = t/n.$$

The number of segments to be retrieved for request R_i in each round would be

$$N_i = r_{c_i} d.$$

Note that with increase in the number of clients being serviced in each interval, the duration of each round decreases. Increasing the number of client requests would reduce the QoS at each client as the data each client can access in each round becomes less and less.

Due to difference in compression techniques of media data and differences in relative placement of segments on the disk, the actual time τ_{ik} taken to retrieve N_i segments in $round_{ik}$ could be different in any two rounds. Vin and co-workers [HGP94] define overflow and underflow rounds if $\tau_{ik} > d$ and $\tau_{ik} < d$ respectively. In an underflow round, the server can read more segments into the buffer so long as the duration of the round does not exceed d. This, termed *read-ahead*, helps in reducing delays in the retrieval of objects in the future rounds. On the other hand, in the case of an overflow round, the server delays the retrieval of the rest of the segments until the next round, i.e., $round_{ik+1}$. Note that this delay need not necessarily reduce the QoS parameters for each client when data of each stream is retrieved before the start of presentation of each stream.

Let N_{ik}^{act} be the actual number of segments read in $round_{ik}$ for request R_i. Given that the average seek time of segments in $round_{ik}$ is $\lambda_{ik} = N_{ik}^{act}/d$, the maximum seek time for a segment belonging to a request R_i can be calculated as

$$\lambda_i^{max} = \max\{\lambda_{ij} \mid 1 \leq j \leq n_r\}, \tag{10}$$

where n_r is the number of rounds completed so far.

4.2 Buffer Management

This section will develop a framework which generates required minimum buffer space for media objects at the server site. The framework guarantees both continuity and synchrony in the presentations at one or more client sites and smooth retrieval and transmission of objects at the server site.

In the above analysis of scheduling multiple clients, multiple requests are simultaneously supported by assuming that, for each request R_i, enough segments of R_i are loaded into memory to allow smooth transmission of the segments to the client. Buffers are needed to satisfy this criterion. It is clear from (1) that the size of buffer at any point in a time interval $[a, b]$ depends on external factors and cannot be predetermined. This

[1] We assume here that the consumption rate is defined to be the number of segments per unit time.

problem may be circumvented by allocating maximum buffer required for each object presented in $[a,b]$. The precise definition of interval would be made clear later.

We will limit the analysis to a case where both the loading function L_{im} and the consuming function C_{im} of segment s_{im} belonging to request R_i are linear. The underlying assumption in the case of the loading function is that the seek time is much less than the load time. Besides, loading of a data stream can be segmented into intervals during which the loading rate can be considered essentially constant. Thus, L_{im} can be approximated as a linear function

$$L_{im}(t, t_{l_s}^{im}) = \begin{cases} 0 & \text{if } t < t_{l_s}^{im}, \\ r_l^{im}(t - t_{l_s}^{im}) & \text{if } t \geq t_{l_s}^{im}. \end{cases} \tag{11}$$

where r_l^{im} is the loading rate of segment s_{im} belonging to request R_i. A linear case of the consuming function is shown in (7). Without loss of generality, let us assume that the loading rate of all segments belonging to client request R_i is constant and that the consumption rate of all segments belonging to request R_i is also constant; that is, $\forall i \; 1 \leq i \leq n_i$,

$$\begin{aligned} r_l^{im} &= r_l^m, \\ r_c^{im} &= r_c^m. \end{aligned} \tag{12}$$

However, segments belonging to different client requests need not have uniform loading or consumption rates. For a detailed analysis of buffer consumption with streams with identical consumption rates, the reader is referred to [NY94].

Clearly, if the loading rate of a segments belonging to request R_i is greater than the consumption rate, all segments belonging to request R_i attain their maximum buffer requirement at the end of loading phase, that is, at $t = t_{l_f}^{im}$, provided $t_{l_f}^{im} \in [a,b]$. In contrast, if the consumption rate is higher than the loading rate, then all segments belonging to request R_i attain their maximum buffer requirement at the beginning of consumption phase, at time $t = t_{c_s}^{im}$, given $t_{c_s}^{im} \in [a,b]$. Let us denote the time at which segment s_{im} belonging to request R_i attains its maximum buffer requirement as t_{max}^{im}. If $t_{max}^{im} \notin [a,b]$, the maximum buffer requirement could occur at one of the bounds of the interval. For the case $r_l^m > r_c^m$, we define the time at which segment s_{im} belonging to request R_i attains its maximum buffer requirement as

$$t_{max}^{im} = \begin{cases} t_{l_f}^{im} & \text{if } a \leq t_{l_f}^{im} \leq b, \\ a & \text{if } t_{l_f}^{im} < a, \\ b & \text{if } t_{l_f}^{im} > b. \end{cases} \tag{13}$$

On the other hand, if $r_c^m > r_l^m$,

$$t_{max}^{im} = \begin{cases} t_{c_s}^{im} & \text{if } a \leq t_{c_s}^{im} \leq b, \\ a & \text{if } t_{c_s}^{im} < a, \\ b & \text{if } t_{c_s}^{im} > b. \end{cases} \tag{14}$$

Given that each segment s_{im} belonging to request R_i needs the maximum buffer space at $t = t_{max}^{im}$ in the interval $[a,b]$, the maximum buffer requirement for all segments loaded in the interval is simply the sum of all maximum buffers over all the objects presented in interval $[a,b]$. Let B^m_{tot} denote the maximum buffer requirement over all the segments belonging to request R_i that are presented in the interval $[a,b]$. It is given by

$$B^m_{tot} = \sum_{i=1}^{n_i} B_{im}(t^{im}_{max}, t^{im}_{c_s}), \tag{15}$$

where n_i is the total number of segments belonging to request R_i in the interval. Therefore, the total buffer requirement over all the requests is

$$\begin{aligned} B_{tot} &= \sum_{j=1}^{n} B^j_{tot}, \\ &= \sum_{j=1}^{n} \sum_{i=1}^{n_j} B_{ij}(t^{ij}_{max}, t^{ij}_{c_s}), \end{aligned} \tag{16}$$

where n is the total number of requests and n_j is the total number of segments belonging to request R_j in the interval.

In a different study [GZ96], the authors have shown the advantage of buffer sharing over the simple buffer management strategy shown above. A full analysis of buffer sharing would involve finding the maximum buffer requirement applicable to all the segments presented in an interval. From the result obtained in [GZ96], we have

Corollary 1 *The maximum buffer requirement over all the segments in interval $[a,b]$ occurs at a time when at least one of the segments attains its maximum buffer requirement in that interval and is given by*

$$B_{shar} = \max\{\sum_{j=1}^{n} \sum_{i=1}^{n_j} B_{ij}(t^{kl}_{max}, t^{ij}_{c_s}) \mid 1 \le l \le n,\ 1 \le k \le n_l\}. \tag{17}$$

Figure 4 shows a typical delivery schedule of two media media streams. The increasing slope refers to the loading of the data from the disk while the decrease relates to the transmission of data from the buffer to the client.

A multimedia server can organize the storage of media data from the disk in terms of fixed size *media blocks* [HGP94] that correspond to the media segments. Thus, the buffer can be segmented into *slots* to hold the media blocks. Without loss of generality, let us assume that the size of media segments is equal to the size of a buffer slot. Let us denote the buffer requirement for retrieving a segment in $round_{ik}$ by $B_{ik}(t, t^{ik}_{c_s})$. Since the disk access rate is usually higher than the transmission rate, the communication manager can start sending data to the client as soon as data is loaded into the buffer by the buffer manager. This also means that the maximum buffer requirement for each media segment would occur at the end of the loading phase in each round. Let t^{ik}_{max} denote the time at which the maximum buffer requirement occurs in $round_{ik}$, and t^s_{ik} and t^f_{ik} as the starting and finishing times of $round_{ik}$ respectively. The maximum buffer requirement would occur at the end of the round, i.e., $t^{ik}_{max} = t^f_{ik}$, and is equal to $B_{ik}(t^f_{ik}, t^{ik}_{c_s})$.

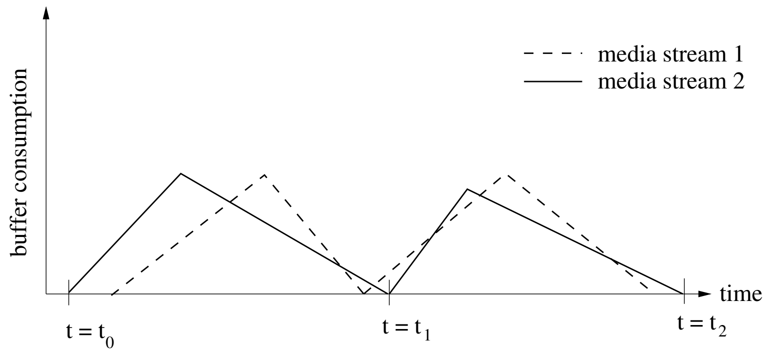

Figure 4: Typical delivery of media streams.

Figure 5 shows the buffer consumption profile for two simultaneous requests. In round 01, the server loads in the data for client request 0, while in round 11 data for client request 1 is loaded. Note that in round 11, whatever is left of the data read for client request 0 in round 01 is also transmitted by the communication manager. Using the buffer consumption model defined above, one can very easily determine the maximum amount of buffer required in each interval. This maximum is then used to define an admission control policy. Note that the maximum buffer requirement in interval 0 in Figure 5 occurs at t_0 and occurs at t_1 in the interval 1.

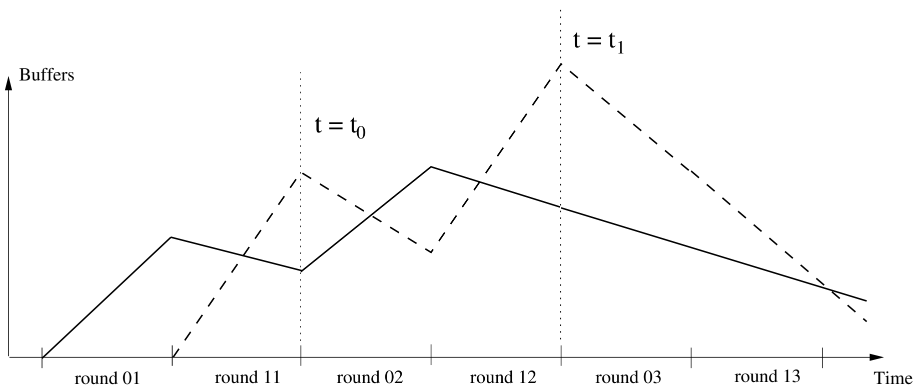

Figure 5: Buffer consumption for two client requests

4.3 Admission Control

Guaranteeing continuous playback for each client request requires that the retrieval time of data in a round does not exceed the playback time of the data retrieved in that round. This necessitates the server to employ admission control to decide whether a new client request can be admitted without violating the deadline requirements of the current client requests. The formulation of an admission control strategy would depend on the QoS requirements of each client and the shared resources at the server site.

Different clients would have varying QoS parameters. The multimedia server should exploit this variation while servicing the client requests. Let α embody the QoS associated with each client request. A strict servicing policy would set α to 1, thereby requiring the server to meet all the deadline requests of the client. A flexible request would have a value in the range [0,1] suggesting that the server can drop some segments if it cannot meet a deadline requirement. In such a case, the server can also delay the transmission of segments so long as the new deadline requirements are within the QoS specifications set by the client. Thus, the fraction of data, such as frames in a video clip, that can be dropped or time through which the server can delay the transmission of segments is set by α.

Another important parameter in admission control is the buffer limitation. Servicing a new request would mean increase in the buffer requirements. Admitting of a new request should not violate the deadline requirements of the current requests being serviced. A simple strategy would be to make sure that the total buffer consumption in next m successive intervals does not exceed the maximum buffer size B_{max}. To do this, one needs an upper bound on the seek times for each request. The upper bounds on the seek times for the current requests can be found using the analysis given in Section 4.1. To find the upper bound on the seek time for a new request, we can find such a bound (say, λ_{max}) by running a calibration program. The minimum number of segments retrieved in a round can be calculated by uniformly spacing the segments across the disk thereby maximizing the seek times [GDN95].

Calculation of m proceeds as follows. Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ denote the set of current requests being serviced. Each request R_i has a maximum seek time λ_i^{max} calculated as described in Section 4.1. Let s_i denote the size of each request $R_i \in \mathcal{R}$ that is left to be read from the disk. The maximum number of rounds that are required for each request $R_i \in \mathcal{R}$ is $\frac{s_i \lambda_i^{max}}{d}$. Note that the value of d is now $t/(n+1)$. The value of m would then be

$$m = \max\{\frac{s_i \lambda_i^{max}}{d} \mid 1 \leq i \leq n\} \tag{18}$$

Adding a new request to the current set of requests would decrease the duration of each round in an interval. Let N_{ik} and $N_{ik}^{'}$ denote the number of segments read for request R_i in $round_{ik}$ when there are n and $n+1$ requests to be serviced respectively. Thus, $N_{ik} - N_{ik}^{'}$ segments are delayed in their retrieval to the next round $round_{ik+1}$. Note that delaying the retrieval to the next round does not necessarily result in a discontinuity in the playback for client request R_i depending on the amount of read-ahead $N_{ik}^{\mathcal{A}}$, which determines the number of segments read from the disk prior to playback initiation [HGP94]. Therefore, the number of segments that can be dropped or delayed without causing a discontinuity is given by the relation

$$N_{ik} - N_{ik}^{'} \ \leq \ N_{ik}^{\mathcal{A}} + N_{ik}(1 - \alpha_i). \tag{19}$$

A simple admission control policy would be a greedy policy that calculates the buffer requirement for each new request and then admits the request if the total buffer consumption in next m successive intervals does not exceed the maximum buffer size B_{max} and the QoS specifications (set by equation 19) of all the current clients requests are not violated. The goal of the admission control algorithm, under such conditions, would be to maximize the number of client requests admitted. The greedy admission control algorithm takes in as input a request $(\{R_1,\ldots,R_n\},R_{n+1})$, that consists of a set of current requests being serviced and the new request R_{n+1} that needs to be admitted, along with the QoS parameter for each client request, and produces as output the decision whether to admit the new request or not.

Greedy admission
Input: $(\{R_1,\ldots,R_n\},R_{n+1})$, B_{max}, α_i $(1 \leq i \leq n+1)$

```
m ← max{s_i λ_i^max / d,  1 ≤ i ≤ n}
for i = 1 to m do
    //Check if QoS specifications of each client is violated
    for k = 1 to n do
        if N_ik − N'_ik > N^A_ik + N_ik(1 − α_i) then return FALSE
    //Calculate the maximum buffer requirement in each interval
    B^i_shar = max{ Σ_{j=1}^{n+1} B_ji(t^{jl}_max, t^{ji}_cs) | 1 ≤ l ≤ n+1 }
    if B^i_shar > B_max then return FALSE
endfor
return TRUE
```

Using buffer sharing does increase the number of client requests admitted over non-shared buffer schemes. This is because more clients can be admitted without overflowing the buffer.

5 Conclusions

We have presented the design and implementation of a client-server distributed multimedia database environment that can be used to support large digital libraries. Several main system architecture and design feature have been provided. In the system design, we have described a data model, QoS specification and a buffer model for supporting both the client and server functionalities. Server functionalities, including client scheduling, data buffering and admission control, are investigated, and the protocols to implement these functionalities are offered. A client request can only be admitted if both the QoS requirements from the client and the upper bound on total buffer consumption at the server are satisfied.

References

[All83] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of ACM*, 26(11), 1983.

[GC92] Jim Gemmell and Stavros Christodoulakis. Principles of delay-sensitive multimedia data storage and retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[GDN95] G.Neufeld, D.Makaroff, and N.Hutchinson. The Design of a Variable Bit Rate Continuous Media Server. Technical Report TR-95-06, Dept. of Computer Science, University of British Columbia, Vancouver, Canada, March 1995.

[GZ96] Sreenivas Gollapudi and Aidong Zhang. Buffer Management in Multimedia Database Systems. In *the Third IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, Hiroshima, Japan, June 1996.

[HGP94] H.M.Vin, A. Goyal, and P.Goyal. Algorithms for Designing Multimedia Servers. In *First IEEE Intl. Conf. Multimedia Computing and Systems (ICMCS'94)*, pages 234–243, Boston, 1994.

[LD93] P. Lougher and D.Shepard. The Design of a Storage Server for Continuous Media. *The Computer Journal*, 36(1):32–42, 1993.

[LG90] Thomas D.C. Little and Arif Ghafoor. Network Considerations for Distributed Multimedia Object Composition and Communication. *IEEE Network Magazine*, pages 32–49, 1990.

[NY94] R. T. Ng and J. Yang. Maximizing Buffer and Disk Utlilizations for News On-Demand. In *Proceedings of the 20th VLDB Conference*, pages 451–462, Santiago, Chile, 1994.

[TK95] Heiko Thimm and Wolfgang Klas. Playout Management – An Integrated Service of a Multimedia Database Management System, 1995. (Technical Report, GMD-IPSI).

[ZG95a] Aidong Zhang and Sreenivas Gollapudi. Multimedia Transaction Management in Database Systems. Technical Report CS-TR-95-44, State University of New York at Buffalo, October 1995.

[ZG95b] Aidong Zhang and Sreenivas Gollapudi. QoS Management in Educational Digital Library Environments. Technical Report CS-TR-95-53, State University of New York at Buffalo, November 1995.