A Multithreaded Client-Server Architecture for Distributed Multimedia Systems

Sreenivas Gollapudi

A Thesis
submitted to the Faculty of the Department of Computer Science
of State University of New York at Buffalo in partial fulfillment
of the requirements for the degree of Master of Science.

July 1996

Acknowledgements

I am greatly indebted to Dr. Aidong Zhang, my advisor, for her invaluable help and guidance throughout this work. I very much enjoyed working with her. This work really helped me better understand the issues involved in developing multimedia systems.

I would like to thank Dr. Sreejit Chakravarty and Dr. Bharat Jayaraman for serving on my thesis committee and commenting on my thesis.

Thanks to Shavinder Multani, D. Sivakumar, and Ajay Nerurkar for helpful discussions and comments during the course of this work. Special thanks to the friendly people in 226 Bell hall - Ellie, Gloria, and Sally. They have been really helpful.

I also take this opportunity to thank my wife, Rama, for her encouragement and support throughout this study.

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 thesis, we present the design and implementation of a client-server distributed multimedia database environment that can be used to support multimedia systems such as large digital libraries. We focus on the problems of playout management and buffer management at the client and buffer management and admission control at the server.

Current database management system techniques are insufficient to support the management of multimedia data owing to their time-sampled nature. The extension of database systems to support multimedia applications thus requires new mechanisms to ensure the synchronized presentation of multimedia data streams. In order to flexibly and efficiently present multimedia data streams to users, media streams must be segmented into media objects with time constraints among these objects specified and maintained. We investigate a framework and systematic strategies for supporting the continuous and synchronized retrieval and presentation of multimedia data streams at the client.

The delivery of continuous and synchronous multimedia data from a database server to multiple destinations over a network presents new challenges in the area of buffer management. We investigate the principles of buffer management for multimedia data presentations in object-oriented database environments. The primary goal is to minimize the response time of multimedia presentations while ensuring that all continuity and synchronization requirements are satisfied. Minimum buffering requirements to guarantee the continuity and synchrony of the presentation of multimedia data are first proposed. A prefetching technique which satisfies these requirements is then developed.

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.

Contents

List of Figures

List of Tables

Chapter 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 maximize utilization of 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 study, 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 distributed multimedia applications 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 chapter is organized as follows. Section 2 introduces the system architecture. In Section 3, we present the system design strategies, including the data model and the transaction model.

1.1 System Architecture

The system architecture under consideration is illustrated in Figure 1.1. This architecture includes a distributed multimedia database management system (multimedia server), a set of multimedia databases, and a set of clients which access the server. The multimedia database management system 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]. A client-server model wherein the client performs the playout management locally is an ideal candidate for implementing the playout management service. The integration of the multimedia playout management and database systems make it possible to efficient interplay between playout management components and database management system components.

As shown in Figure 1.1, the distributed multimedia database management system server contains two main modules: a multimedia task language (MTL) interpreter and a media manager (MM). The multimedia task language MTL interpreter allows users to specify a set of tasks associated with a multimedia task, including intra- and inter-synchronization requirements on component tasks. A

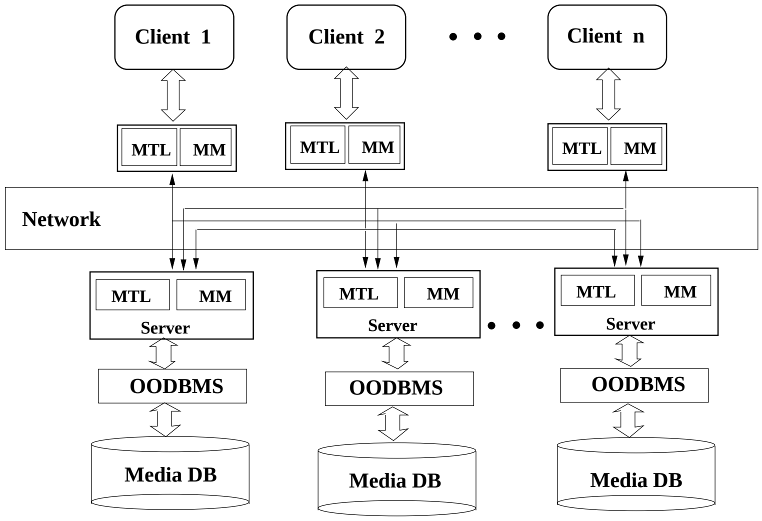

Figure 1.1: System architecture

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.

1.1.1 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 an object-oriented database system. The server is an event driven process that uses a time-ordered prioritized query queue to service requests[GZ96b]. Figure 1.2 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 client has to read data at the same rate as the transfer rate to minimize the buffer requirements at the client. The onus of interpretation of the media data falls on the client as data is presented at the client workstation. At present, two types of data, digital video and audio are supported.

The server has three components, *viz* 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

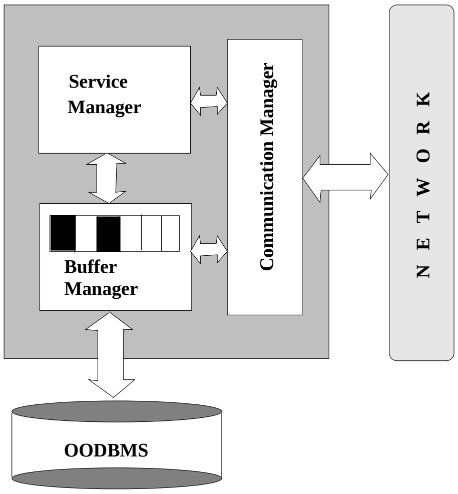

Figure 1.2: Server configuration

requirements of clients. The communication manager is multi-threaded. 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. Each module is described in more detail in Chapter 6. The handling of multiple client requests and buffer management are detailed in chapter 5.

The server and the clients exchange information via a virtual circuit connection (TCP). At present the server and the clients sit on a LAN. The client controls the rate at which the server must transmit data to the client. The client sets the rate at the beginning of each connection and can change it at the beginning of a new request.

1.1.2 Multimedia Client

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

The 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 presented. The scheduler is responsible for synchronizing the delivery of 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

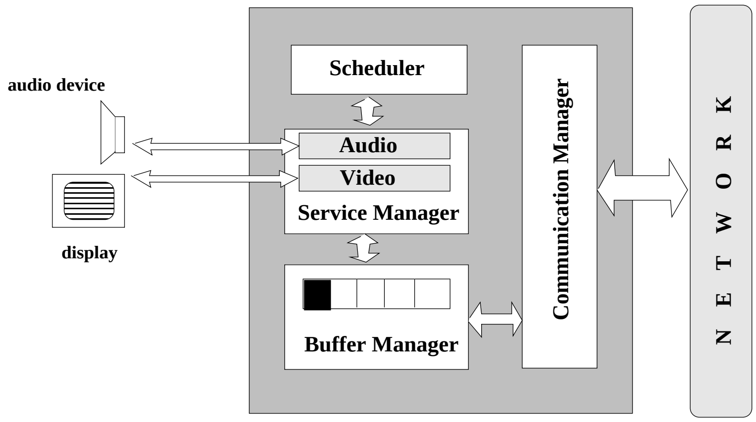

Figure 1.3: Client configuration

is the basis for buffer management at the client and server sites is presented in the chapter 3. Figure 1.3 shows the logical layout of the client. Both scheduling and buffer management at the client site will be discussed in chapters 2 and 3 respectively.

1.2 Multimedia Transaction 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 object). It is then dissected into a set of semantically meaningful media objects which may be further classified into several higher-level object classes. Objects from different media streams may also be spatiotemporally 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 transaction management level for the purpose of scheduling media data operations.

1.2.1 Data Model

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 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 $\langle o, t, \triangle t \rangle$ to denote that object o is to be delivered at time t and will last time period $\triangle t$.

Media objects from different data streams may need to be linked through time constraints to specify their synchronization; such time constraints are termed inter-synchronization requirements. For example, in slide presentation applications, an audio object must be played along with a slide object. The temporal relationship between two atomic objects from different media streams may also conform to any of the thirteen temporal relationships described in [All83]. Inter-synchronization requirements may be specified as meta-data or specified in transaction programs. In some cases, the relative time and time interval associated with an atomic object may need to be adjusted to conform with these inter-synchronization requirements. Such synchronization requirements within and between media streams are discussed in more detail in Chapter 2.

1.2.2 Multimedia Transactions and Schedules

We will now discuss the proposed multimedia transaction model. Since our primary concern with multimedia data involves retrieval rather than update, our model will consider only delivery operations of atomic objects. We shall now introduce the concept of a *transaction*. For the elements of a transaction, we assume the availability of three basic operations: $start(t)$, $end(t)$, and $deliver(o, t)$, where $start(t)$ and $end(t)$ are beginning and termination operations at a relative time t, and $deliver(o, t)$ is a delivery operation of object o at relative time t. A *transaction* is then defined as a partial order of start, end, and delivery operations which contain exactly one start operation that is the minimum (first) element in the partial order, one end operation that is the maximum (last) element in the partial order, and all delivery operations executed on a given data stream. A *multimedia transaction* consists of a set of transactions upon which synchronization constraints are specified on the delivery operations to enforce both intra- and inter-synchronization requirements.

We define a schedule to be an execution of a multimedia transaction. A schedule consists of a set of sub-schedules, each of which is an execution of a transaction within the multimedia transaction. We define a *synchronization point* to be a point held in common by delivery operations from all participating transactions needing to be synchronized. Synchronization points define the junctures at which sub-schedules must be synchronized. A scheduler must ensure the correct execution of a multimedia transaction. A schedule or sub-schedule may differ from a multimedia or individual transaction because the dynamic time constraints of the former differ from the static relative time assignments of the latter. Additionally, the tolerance parameters given for a schedule will permit further deviations from the multimedia transaction.

The remainder of this thesis is organized as follows. Chapter 2 introduces an effective approach to specify synchronization dependencies on multimedia transactions. We then present the scheduling principles and algorithms to ensure the synchronous presentation of media streams in the event of delays. In chapter 3 the principles of buffer management for individual media streams, both with or without delays are discussed. We then present a buffer consumption model for multiple media streams forms the basis for a buffer allocation and replacement strategy. In chapter 4, we discuss QoS parameters and introduce the correctness criteria to preserve the quality of media presentations. Chapter 5 investigates the approaches for the server to schedule client requests, maintain upper bound on buffer consumption and control the number of client requests to be admitted for service by the server. Chapter 6 sets forth our implementation and experimental results. Concluding remarks and future work are offered in Chapter 7.

Chapter 2

Scheduling in Multimedia Systems

Multimedia data refers to the simultaneous use of data in different media forms, including images, audio, video, text, and numerical data. 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. Such synchronization requirements are generally specified by either spatial or temporal relationships among multiple data streams. For example, a motion video and its caption must be synchronized spatially at the appropriate position in a movie, and, in a slide presentation, a sequence of images and speech fragments must be temporally combined and presented to compose one unified and meaningful data stream. This research will focus on only temporal synchronization.

Current database systems are not equipped to represent the entire multimedia data flow, nor may it be desirable for them to support the plethora of retrieval types required to support multimedia data. Thus, continuous media data must be parsed into representable segments, which we term *media objects*. In order to re-present the original data stream to users, synchronization constraints among media objects both within and between media streams must be specified and maintained. Such synchronizations are usually termed *intra-synchronization* and *inter-synchronization* respectively. For these reasons, synchronization has been recognized as one of the central problems in multimedia system development [LG90b, Ste90].

2.1 Previous Work

Substantial research has been directed toward the support of synchronization within operating systems and network architectures [RV93, Ste90, AH91, RRK93, GR93, ZF93]. Through this research, new behavioral concepts required for multimedia data have been identified and mechanisms have been proposed to enhance such conventional storage, synchronization, and communication mechanisms as random disk allocation, semaphores, monitors, or RPC. However, the techniques proposed at these levels are insufficient to address the problems encountered at the transaction management level in database systems. For example, Anderson et al. [AH91] describe techniques for recovering from loss of synchronization between interrupt-driven media I/O devices, and Rangan et al. [RRK93] devise techniques for inter-media synchronization during on-demand multimedia retrieval from a server to multiple destinations over integrated networks. Both research efforts are directed toward the synchronization of two types of media streams: master and slave streams. In cases of asynchrony, synchrony is restored by deleting (skipping) media units of a slower slave by the amount it lags behind the master, or by duplicating (pausing) media units of a faster slave by the amount by which it leads the master. Consequently, slower slaves catch up with the master, and faster slaves are forced to wait for the master. However, in the transaction management level, a generic scheduling strategy is needed to enforce the synchronization constraints on multiple media streams. Furthermore, several parameters such as average delay, speed ratio, utilization, jitter, and skew [LG90a] must be considered in scheduling in order to offer effective synchronization maintenance for the presentation of multiple media streams.

In the proposed framework, we use the multimedia data model described in Chapter 1 for the purpose of scheduling media data operations. This data model facilitates the specification of time constraints on media objects. A multimedia transaction defines a synchronous presentation of media streams and a schedule is an execution of a multimedia transaction. A schedule may differ from a multimedia transaction because the dynamic time constraints of the former differ from the static relative time assignments of the latter. An approach to the specification of synchronization constraints among the component transactions of a multimedia transaction is offered. These specifications provide a vehicle for the scheduler to ensure the synchronous execution of these component transactions. We then investigate the principles guiding the scheduling of the delivery of multiple media data streams. Scheduling strategies are developed to permit efficient resynchronization of the presentation of multiple media streams in the event of delays. Parameters including average delay, speed ratio, utilization, jitter, and skew are integrated in the scheduling algorithms. Our implementation and experimental results based on the proposed approaches are described in Chapter 6.

2.2 Realization and Specification of Synchronization Constraints

In this section, we will propose an approach to the specification of synchronization constraints among the component transactions of a multimedia transaction. Through these specifications, a scheduler can ensure the synchronous execution of these component transactions.

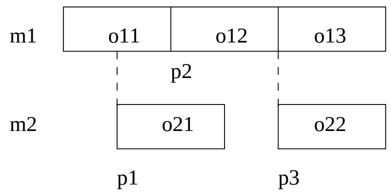

Figure 2.1: Synchronized streams.

2.2.1 Ensuring Time Constraints

As indicated in the multimedia transaction model given in the previous chapter, both data and transaction operations are associated with time constraints. Synchronization constraints may also exist among the component transactions of a multimedia transaction. Since synchronization constraints are implicitly imposed by the specification of time constraints, the maintenance of the latter also guarantees the maintenance of the former. Synchronization constraints therefore need not be explicitly specified and enforced by the scheduler. However, while the scheduler should make every effort to enforce the time constraints defined on transaction operations, even minor delays may create great difficulties in scheduling. The following example is illustrative:

Example 1 *Consider two media streams m_1 and m_2 which must be synchronized in delivery. Let m_1 consist of $\langle o_{11}, 0, 1\rangle$, $\langle o_{12}, 1, 1\rangle$, $\langle o_{13}, 2, 1\rangle$ and m_2 consist of $\langle o_{21}, 0.5, 1\rangle$, $\langle o_{22}, 2, 1\rangle$. Assume that multimedia transaction T consists of two transactions t_1 and t_2 which represent m_1 and m_2, respectively. The synchronization constraints between t_1 and t_2 are illustrated in Figure 2.1. If time constraints only are enforced, then the delay of any object will result in confusion in the synchronization among ensuing objects.*

Since experimental experience demonstrates that such delays are frequent, the explicit specification and enforcement of synchronization constraints is necessary.

2.2.2 Generating Synchronization Dependencies

Synchronization dependencies among the delivery operations in a multimedia transaction are dynamically generated on the basis of the intra- and inter-synchronization constraints placed on the media streams. Such dependencies are intended to facilitate scheduling by efficiently describing the synchronization constraints existing among the transactions of each multimedia transaction.

Let a multimedia transaction be defined as a set of transactions $t_1, .., t_n$ which represent the synchronized presentation of media streams $m_1, ..., m_n$. Each media stream m_i $(1 \leq i \leq n)$ consists of a set of objects and each object is specified as $\langle o_{ij}, t_{ij}, \triangle t_{ij}\rangle$ $(1 \leq j \leq i_m)$. Without loss of generality, we assume that the synchronization constraints are implied in the definition of time

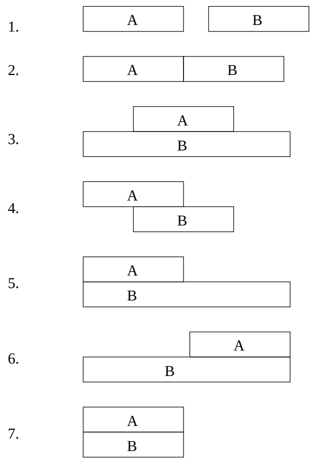

Figure 2.2: Typical temporal relationships.

constraints on objects. The synchronization relationship between any two objects in either a single media stream or two media streams follows the thirteen temporal relationships outlined in [All83]; these are given in Figure 2.2. Inverse relationships are not listed in the figure. In general, these synchronization relationships can be easily generalized to the synchronization relationships among n media objects [DDI$^+$95].

Three relationships, namely *before*, *after*, and *equal*, are used to define the temporal ordering of one object with respect to another. The temporal relationships between two objects o_i and o_j are categorized as: (1) o_i is said to start *before* o_j if $t_i < t_j$; (2) o_i is said to start *after* o_j if $t_i > t_j$; (3) o_i is said to start *equal* to o_j if $t_i = t_j$; (4) o_i is said to end *before* o_j if $t_i + \triangle t_i < t_j + \triangle t_j$; (5) o_i is said to end *after* o_j if $t_i + \triangle t_i > t_j + \triangle t_j$; (6) o_i is said to end *equal* to o_j if $t_i + \triangle t_i = t_j + \triangle t_j$.

We associate each media object $\langle o_{ij}, t_{ij}, \triangle t_{ij} \rangle$ with two events: a START event, denoted by $s_{o_{ij}}$, and an END event, denoted by $e_{o_{ij}}$. We say that two objects $\langle o_{i1}, t_{i1}, \triangle t_{i1} \rangle$ and $\langle o_{i2}, t_{i2}, \triangle t_{i2} \rangle$ with o_{i1} starts before o_{i2} in a single media stream are *neighboring* each other if there is no object $\langle o_{i3}, t_{i3}, \triangle t_{i3} \rangle$ such that $t_{i1} + \triangle t_{i1} \leq t_{i3} + \triangle t_{i3} \leq t_{i2}$. We introduce three types of synchronization points within media streams, as follows:

Definition 1 (Intra-synchronization point) *A media stream m_1 has an intra-synchronization point p_1 if there exist two neighboring objects $\langle o_{1i}, t_{1i}, \triangle t_{1i} \rangle$ and $\langle o_{1i+1}, t_{1i+1}, \triangle t_{1i+1} \rangle$ such that $s_{o_{1i+1}} = e_{o_{1i}} + \triangle t$, where $\triangle t = t_{1i+1} - (t_{1i} + \triangle t_{1i})$.*

Definition 2 (Inter-synchronization point) *A media stream m_1 has an inter-synchronization point p_1 if there exists an object o_{1i} in m_1 and another media object $\langle o_{jk}, t_{jk}, \triangle t_{jk} \rangle$ (may or may not be in m_1) such that o_{1i} starts equal to o_{jk} or o_{1i} ends equal to o_{jk}.*

Definition 3 (Middle-synchronization point) *A media stream m_1 with object $\langle o_{1i}, t_{1i}, \triangle t_{1i} \rangle$*

has a middle-synchronization point p_1 *if there exists another media object* $\langle o_{jk}, t_{jk}, \triangle t_{jk} \rangle$ *(may or may not be in* m_1*) such that* $t_{1i} < t_{jk} < t_{1i} + \triangle t_{1i}$ *or* $t_{1i} < t_{jk} + \triangle t_{jk} < t_{1i} + \triangle t_{1i}$.

In Example 1, p_1 is a middle-synchronization point, p_2 is an intra-synchronization point, and p_3 is both an intra-synchronization and inter-synchronization point. The synchronization relationships given in Figure 2.2 can also be categorized into three classes, according to their synchronization points: cases 1 and 2 have intra-synchronization points; cases 5, 6, and 7 have inter-synchronization points; and cases 3, 4, 5, and 6 have middle-synchronization points.

We define the granularity of a media object to be the size of the object and the granularity of the synchronization between a set of media streams to be the number of synchronization points that must be identified. Clearly, the finer the object granularity, the more synchronization points will need to be identified. Thus, the design of a higher-level data model for the decomposition of media objects determines the minimum granularity of the synchronization between the media streams. However, at the level of multimedia transaction management, the granularity of the synchronization can be defined more finely. At this level, additional synchronization points can be defined in the midst of objects to permit finer synchronization control among media streams. As the decomposition of media objects is not the main concern of this paper, this subject will not be discussed further at this point.

All START and END events are then classified into layered GROUPs based on the time constraints pertaining to the events. The lowest-layer GROUP_1 contains all START events which are the events at the starting time of the entire presentation, and the highest-layer GROUP_n contains all END events which are the events at the ending time of the entire presentation. All events occurring at a given time belong to the same GROUP. Thus, each GROUP contains all START and END events that must be simultaneously executed. Within each GROUP, all the END events are related to the START events by the *before* relationship. Events between two consecutive groups are related by the *after* relationship.

At each middle-synchronization point, the object o_{ij} to which there are other objects that start or end in the middle will be split by assigning a START and an END event at the middle-synchronization point.

The following example demonstrates an application which uses the specification described above.

Example 2 *Consider an application involving on-line computer-assisted learning in undergraduate education. Without loss of generality, we assume that there are two media streams, audio and slides (or video), in each multimedia transaction. Intra-synchronization within the slide stream may require that two objects either overlap or be sequentialized. Intra-synchronization within the audio stream requires only that objects be sequentialized. Additional inter-synchronization requirements between the two media streams are specified among slides and audio objects. These requirements*

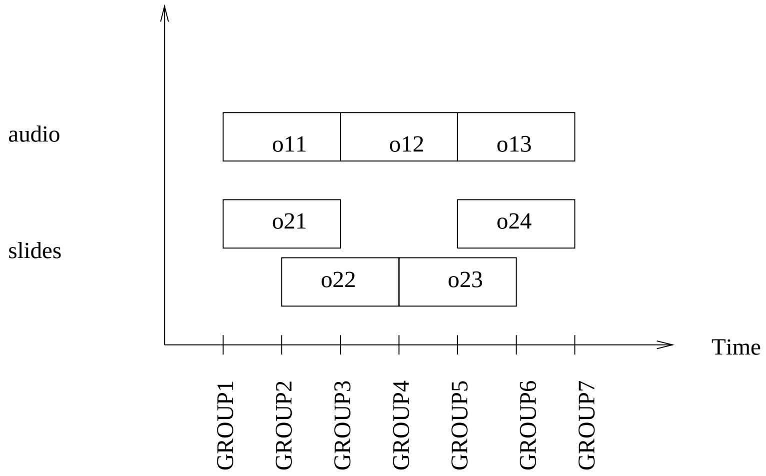

Figure 2.3: Layered GROUPs.

between the slides and audio objects follow the thirteen temporal relationships outlined in [All83]. Let a multimedia transaction contain two transactions, one of which accesses the slide stream and the other the audio stream. In order to successfully deliver both streams to a student, the system must ensure that all time constraints placed on the individual delivery operations and the synchronization between slides and audio objects are preserved. Let a particular application is given in Figure 2.3. In this application, a set of layered GROUPs, denoted $GROUP_1$*,...,* $GROUP_7$*, are identified.*

2.3 Principles of Scheduling

In this section, we will investigate the principles guiding the scheduling of the delivery of multiple media data streams. We shall assume that the given transportation and buffer management schemes provide sufficient support for delivering media objects. A framework will be developed to permit efficient synchronization of the presentation of multiple media streams.

In our context, the scheduling of multimedia transactions includes the scheduling of time-dependent delivery operations, synchronized delivery enforcement among multiple transactions of a multimedia transaction, and delivery delay recovery. The functionality of multimedia transaction scheduling is similar to that of conventional concurrency control and failure recovery. However, while conventional concurrency control and recovery schemes enforce a consistency requirement, a correctness criterion in this instance must verify that delivery operations are performed according to a predefined synchronization pace and within the time constraints imposed on transactions. Real-time concurrency control has been an active research topic for several years and deals with similar issues [AGM88, BMHD89, ZRS87a, ZRS87b, SRL88, DBB+88, TCG+93]. However, the issues which arise in a multimedia context differ from those in real-time concurrency control, since the preservation of database consistency is not of primary concern to the continuous and synchronized delivery of media streams. Furthermore, although real-time concurrency control handles time-dependent operations, these are primarily concerned with guaranteeing the real-time behavior of transactions. In contrast, multimedia transaction scheduling must provide support for the synchronization of media operations. Several additional important parameters which are not considered in real-time concurrency control must be factored into multimedia transaction scheduling.

We will first define a correctness criterion for the execution of a multimedia transaction and then identify those schedules to be considered to be correct. As with conventional transactions, the semantics of a multimedia transaction determine the correctness of its execution. Unlike conventional transactions, however, the time constraints defined within multimedia transactions assume a position of prime importance. We thus introduce the following semantic correctness criterion:

Definition 4 (Semantic correctness criterion) *The execution of a multimedia transaction T is correct if the time constraints specified on the transactions in T are preserved.*

This semantic correctness criterion is theoretically applicable to the executions of multimedia transactions. However, in a practical, delay-prone system, this criterion cannot be applied directly by the scheduler to enforce the execution of multimedia transactions. Given the pervasive nature of delays, a strict application of this rule would result in the aborting of the vast majority of multimedia transactions. A more realistic scheduling criterion is therefore needed. The discussion of the specification of multimedia transactions provided in the previous section indicates that the threefold categorization of synchronization points summarizes the most critical scheduling information. Additional synchronization points could be specified between these synchronization points within the shared intervals. While a finer granularity of stream synchronization improves the synchronization of media stream presentation, it also increases the control-related scheduling overhead. We identify *synchronous schedules* below which use the three types of synchronization points defined earlier.

Definition 5 (Synchronous schedules) *A schedule S of a multimedia transaction is synchronous if all intra-, inter-, and middle-synchronization points defined among the transactions of the multimedia transaction are preserved.*

Thus, synchronous schedules guarantee that no temporal deviation will occur within the simultaneous presentation at synchronization points. Since delays that may occur between these synchronization points are not considered, a synchronous schedule may actually fail to preserve the time constraints defined on the delivery operations within transactions. There may therefore be a temporal deviation between the delivery operations of different transactions during these intervals. However, any asynchronization caused by delays will be recovered at the next synchronization point.

Thus, synchronous schedules preserve all the synchronization constraints defined on transactions by enforcing synchronization at synchronization points.

A synchronous schedule may allow enormous delays between the delivery operations of different transactions during some intervals. Thus, synchronous schedules should not be considered to be completely correct. We introduce the concept of *acceptable* schedules by incorporating the effect of delays into the definition of synchronous schedules.

Definition 6 (Acceptable schedules) *A schedule S is acceptable if and only if S is synchronous and all delays occurring between synchronization points are within the permissible range.*

Note that the scheduler need not consider the concurrent execution of multiple multimedia transactions. There is no need to have a central scheduler to manage the executions of all multimedia transactions, since no updates are involved. An individual scheduler can control the presentation of multiple media streams to a single user.

2.4 Basic Scheduling Strategy

We will now investigate the principles guiding the scheduling of the delivery of multiple media data streams at each client site. We shall assume that the given transmission and the server provide sufficient support for delivering media objects.

We associate each media object $\langle o_{ij}, t_{ij}, \triangle t_{ij} \rangle$ with two events: a START event, denoted by $s_{o_{ij}}$, and an END event, denoted by $e_{o_{ij}}$. All START and END events are then classified into layered GROUPs based on the time constraints pertaining to the events. The lowest-layer GROUP$_1$ contains all START events which are the events at the starting time of the entire presentation, and the highest-layer GROUP$_n$ contains all END events which are the events at the ending time of the entire presentation. All events occurring at a given time belong to the same GROUP. Thus, each GROUP contains all START and END events that must be simultaneously executed. Within each GROUP, all the END events and the START events are related by the *before* relationship. Events between two consecutive groups are related by the *after* relationship.

At each middle-synchronization point, the object o_{ij} to which there are other objects that start or end in the middle will be split by assigning a START and an END event at the middle-synchronization point.

We will now discuss the generation of acceptable schedules. Following Definitions 1, 2, and 3, at each intra-synchronization point, there exists an END event and a START event; at each inter-synchronization point, there is one START event; at each middle-synchronization point, there may exist either one START event or an END event and a START event. Synchronization points of all three types can coexist at a given synchronization point; in such a case, the synchronization point must be ensured to be both continuous and synchronous with other specified synchronization points in different media streams.

The scheduler ensures that only synchronous schedules will be generated by controlling the invocation order of events in the formulated layered GROUPs of each multimedia transaction. Let a multimedia transaction T have n layered GROUPs. Assuming no distortion, the basic invocation policy for the execution of T is as follows: assume that all START events in GROUP$_1$ have been invoked.

(1) The events in GROUP$_{i-1}$ always have a higher invocation priority than those in GROUP$_i$, for any i such that $1 < i \leq n$.

(2) All START events in a GROUP$_i$ ($1 \leq i \leq n$) are invoked simultaneously.

(3) All END events in a GROUP$_i$ ($1 \leq i \leq n$) are terminated simultaneously.

(4) All START events in a GROUP$_i$ ($1 \leq i \leq n$) can only be invoked after all END events in the same GROUP have terminated.

Items (1) and (4) ensure the intra- and middle-synchronization points specified in T, while items (2) and (3) ensure the inter- and middle-synchronization points specified in T.

The effects of a variety of delays, including network delays and buffer delays, are not considered in the above policy. As was noted earlier, such effects must be incorporated into the scheduling policy to generate acceptable schedules. In this section, we incorporate the effect of delays into the scheduling policy by propagating delays into the invocation of successive delivery operations. Thus, if a delivery operation is delayed, then the END event in the corresponding GROUP will be delayed and, consequently, all END and START events in the same GROUP will be delayed. By propagation, all the ensuing delivery operations and events in the higher-layer GROUPs will be delayed. A full consideration of delay recovery will be presented in the next section.

There must be a permissible delay constraint that defines the maximum tolerable delay during the execution of multimedia transaction T. We assume that each media stream has a permissible delay constraint and the minimum value of all permissible delay constraints given in the media streams defines the maximum tolerable delay for the multimedia transaction. If a larger delay occurs, then *timeout* will be used by scheduler. If the scheduler finds that it has been waiting too long for the completion of a delivery operation, then it aborts the execution of the multimedia transaction. The effects of delays and timeout actions is incorporated in Algorithm 1.

Algorithm 1 (Basic scheduling):
Input: T, the multimedia transaction with l number of layered GROUPs.
Coordinator for a multimedia transaction
    $i \leftarrow 1$
    invoke All START events in GROUP$_i$
    for $i = 2$ **to** l **do**
        if there exist END events in GROUP$_i$ whose START events are in GROUP$_1$ to GROUP$_{i-1}$
        then wait for the END events to complete
                on timeout begin abort T; **return end**
        /* initiate START events after all END events are complete */
        if there exist START events in GROUP$_i$
        then invoke all START events in GROUP$_i$
    endfor
    terminate all Participants
return
Participant for an individual transaction
    do
        if receive START signal from Coordinator
            execute delivery operation
            send END signal to Coordinator
        else receive TERMINATE signal from Coordinator
            terminate; **return**
    forever

Clearly, Algorithm 1 enforces all defined synchronization points by controling the invocation of START and END events. In addition, timeout is used in case a delivery operation is delayed beyond the permittable delay. Thus, Algorithm 1 generates only acceptable schedules.

2.5 Scheduling with Delay Recovery

In this section, we will investigate the principles involved in the scheduling of multimedia transactions with delay recovery. A framework will be developed to permit efficient resynchronization of the presentation of multiple media streams in the event of delays.

In the previous section, a simple solution was presented in which delays are simply propagated to the ensuing delivery operations. We will now systematically investigate a novel and more effective delay recovery approach. Little and Ghafoor [LG90a] have proposed several parameters to measure the Quality of Service (QOS) for multimedia data presentation. The following parameters have been listed: (1) average delay, (2) speed ratio, (3) utilization, (4) jitter, and (5) skew. A detailed discussion of these parameters will be taken up in Chapter 4. Ideally, both speed ratio and utilization should equal 1. 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.

In the discussion presented in the previous section, our primary goal has been to avoid any deviations from the synchronization constraints associated with the media streams. Thus, Algorithm 1 was primarily concerned with minimizing the possibility of jitter and skew. In fact, the conducted experiments given in Chapter 6 demonstrate that both parameters are close to zero. Our discussion of delay recovery will consider not only the constraints of synchronization but also the parameters of average delay, speed ratio, and utilization.

While the delivery of each media stream would ideally minimize the average delay and maintain the parameters of speed ratio and utilization to be close to 1, the achievement of these three goals is actually in conflict. There must therefore be tradeoffs 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 then, in case that an audio object is delayed, drop some video frames in the corresponding video object to maintain synchronization between the two objects. If the scheduler tries to maintain 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.

In our approach, two parameters, *maximum delay*, denoted $\triangle d_i^{max}$, and *maximum skip*, denoted $\triangle s_i^{max}$, can be specified for each media stream m_i. These two parameters provide users with flexibility in achieving the above goals. If maintenance of good utilization is of highest interest in a particular instance, then the amount that can be skipped should be specified as a relatively small figure. Conversely, If it is more important to minimize the average delays, then the delay allowed for the media stream should be set at a relatively low level. Normally, $\triangle d_i^{max}$ is much larger than $\triangle s_i^{max}$. Under these conditions, our approach will maximize utilization and minimize the abortion rate of multimedia transactions in order to preserve the quality of the presentations.

Consider a synchronous presentation of media streams $m_1, ..., m_n$ and a set of media objects from these streams are currently synchronously delivered. Let $\triangle d_i$ $(1 \leq i \leq n)$ denote the delay that is occurred in the object belonging to m_i. We may have the following situations for these media objects:

(1) $\triangle d_i \leq \triangle s_i^{max}$ for all integer i in the range $1 \leq i \leq n$.

(2) For all i $(1 \leq i \leq n)$, $\triangle d_i - \triangle s_i^{max} \leq min(\{d_1^{max}, d_2^{max}, ..., d_n^{max}\})$ and there exists some $\triangle d_i$ $(1 \leq i \leq n)$ such that $\triangle s_i^{max} < \triangle d_i$.

(3) There exists some $\triangle d_i$ ($1 \leq i \leq n$) such that $\triangle d_i - \triangle s_i^{max} > min(\{d_1^{max}, d_2^{max}, ..., d_n^{max}\})$.

In case (1), synchronous presentation can be restored by simply skipping by the interval by which the delayed media streams lag behind. In case (2), since there exists some media object such that its delay is larger than its permittable skip, simply skipping the delayed objects may not be applicable. However, this difficulty can be circumvented by a compromise between skipping and pausing. Similar to the situation discussed in the previous section, we assume that the timeout period is the minimum value of all permittable delay constraints given in the media streams belonging to the multimedia transaction. Within the permittable timeout period, we calculate the maximum difference between the delay and the allowable skip for delayed objects:

$$\text{PAUSE}= max(\{\triangle d_i - \triangle s_i^{max} | i = 1, ..., n\}). \quad \text{(a)}$$

If the amount of PAUSE is less than the permittable timeout period, then the period of PAUSE will be paused in order for those delayed operations to catch up for the maximum possible period. After this PAUSE period, if there exists some unfinished operation, then it must be within its permittable skip period. Therefore, such an operation can be skipped.

Thus, in case of delays during the execution of GROUP_{i-1}, the following rule is added to the basic invocation policy given in the previous section:

(5) Pause the START events in GROUP_i for a period defined in (a) before invocation.

In case (3), the execution of the multimedia transaction must be aborted. The detailed algorithm of this approach is offered below. Note that the calculation of WAIT in Algorithm 2 is slightly different from PAUSE, since the first completed END event might be delayed and this delay effect must be added to WAIT while performing tolerance check.

2.6 Effect of Buffer Management on Scheduling

Buffer management is essential to ensure the maintenance of the intra-synchronization of a media stream presentation. To facilitate a hiccup-free presentation, we must ensure that an object is present in memory before it is delivered. If the loading rate of a media stream from disk to memory is less than the delivery rate of the media stream, preloading of the stream prior to delivery would be necessary to ensure continuous presentation. Furthermore, an appropriate allocation and replacement strategy must be provided to anticipate the demands of delays and user interactions. Such a strategy must minimize the response time of media presentations while guaranteeing that all continuity and synchronization requirements are satisfied. Thus, the central issues in this context involve the provision of efficient preloading and replacement mechanisms for caching media objects at each client site. The following chapter details a buffer model and a prefetching strategy based on the model.

Algorithm 2 (Scheduling with delay recovery):
Input: multimedia task T with l number of layered GROUPs; allowable SKIP and DELAY
for media objects in T.

```
Coordinator for a multimedia task
    i ← 1
    invoke All START events in GROUP_i
    for i = 2 to l do
        if there exist END events in GROUP_i whose START events are in GROUP_1 to GROUP_{i-1}
        then wait for e ← the first END event to complete
                on timeout begin abort T; return end;
            for the rest of END events do
                WAIT ← the maximum difference between the current delay and SKIP among
                    the delayed media objects
            /* delay of e would be 0 if the first END event is not delayed */
            if WAIT > 0 and WAIT+ the delay of e ≤ min({d_1^max, d_2^max, ..., d_l^max})
            then wait for WAIT
                send STOP signals to all delayed END events in GROUP_i;
            elseif WAIT+ the delay of e > min({d_1^max, d_2^max, ..., d_l^max})
                begin abort T; return end;
        if there exist START events in GROUP_i
        then invoke all START events in the GROUP_i.
    endfor
    terminate all Participants
    return
Participant for an individual task
    do
        if receive START signal from Coordinator
            execute delivery operation
            send END signal to Coordinator
        elseif receive STOP signal from Coordinator
            terminate delivery operation
        elseif receive TERMINATE signal from Coordinator
            terminate; return
    forever
```

Chapter 3

Buffer Management in Multimedia Systems

Buffer management within the multimedia database system is essential to ensure the maintenance of the intra- and inter-stream synchronization requirements of multimedia data presentations. If multiple media streams are synchronously presented, the buffer space must be sufficient to simultaneously hold all these streams. Furthermore, an appropriate allocation and replacement strategy must be provided to anticipate the demands of delays and user interactions. Such a strategy must minimize the response time of multimedia presentations while guaranteeing that all continuity and synchronization requirements are satisfied. Thus, the central issues in this context involve the provision of efficient preloading and replacement mechanisms for caching media objects.

3.1 Previous Work

Research involving buffer management in multimedia database systems is still in its infancy [TK95]. Moser et al [MKK95] have proposed a buffer strategy termed "least/most relevant for presentation." This buffer strategy investigates the effects of such user interactions as "rewind" and "fast forward" on buffer design. A mechanism is proposed which reduces the ensuing delay after user interactions. Chaudhuri et al [CGS95] have investigated the problem of continuously displaying composite objects that are dynamically specified at the server level. Techniques based on simple sliding and buffered sliding are proposed which support continuous display by partial prefetching of overlapping media objects. Such an approach is preferable to the naive strategy of prefetching the entirety of overlapped media objects. Ng and Yang [NY94] have attempted to maximize buffer and disk utilization within the constraints of a fixed amount of buffer space and disk bandwidth, both of which are pre-determined in the design phase. Prefetching strategies that are applicable to database systems are proposed which maximize the throughput of the system.

In this paper, we systematically study the principles underlying buffer management for multimedia data presentation in object-oriented database environments. Media streams are stored as sets of media objects coupled with specified synchronization requirements. Complementary to the work proposed by Chaudhuri et al [CGS95], we investigate the minimum buffering requirements that are necessary to guarantee the continuity and synchrony of the presentation of multimedia data. This is accomplished by first determining the minimum consuming times of media objects following their loading times, while guaranteeing the hiccup-free presentation of the media streams. The presentation of both a single media stream and multiple streams with or without the effect of delays are considered. We then study the principles of media objects consumption. Least upper bounds on buffer sharing among media objects are defined. These minimum buffering requirements establish a framework for buffer management in the multimedia database system. A prefetching technique which satisfies the minimum buffering requirements will be developed. These principles and technique provide users with the full range of information required to develop a multimedia data presentation environment.

3.2 Buffer Model

In this section, we will present a framework which generates the required loading times for media objects and start of consumption times for media streams. This framework guarantees the continuity of media stream presentation while minimizing buffer utilization at both client and server sites. Furthermore, an appropriate allocation and replacement strategy must be provided to anticipate the demands of delays and user interactions.

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. 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.

3.2.1 Consumption times of Media Streams

Normally, the loading of a media stream from disk to memory is much faster than the consumption of the media stream. However, both network and storage delays must be considered in determining the consumption time of a media stream. We first consider the situation of the entire stream. Let $t^m_{l_s}$ be the time at which the loading of media stream m begins and *loading function* $L_m(t, t^m_{l_s})$ be the size of media objects of m read at time t. Let $t^m_{c_s}$ be the time at which the consumption of data stream m begins and *consuming function* $C_m(t, t^m_{c_s})$ be the size of media objects consumed at time t. The size of media objects that must be buffered at any given time is then given by

$$B_m(t, t^m_{c_s}) = \begin{cases} 0 & \text{if } t < t^m_{l_s}, \\ L_m(t, t^m_{l_s}) & \text{if } t^m_{l_s} \le t < t^m_{c_s}, \\ L_m(t, t^m_{l_s}) - C_m(t, t^m_{c_s}) & \text{if } t \ge t^m_{c_s}. \end{cases} \tag{3.1}$$

Delays may be caused by network delays or physical storage delays. In case that the presentation of a media stream is delayed, some data that are expected to be already delivered may still remain in the server buffer or disk. We assume that a parameter, *maximum permitted delay*, denoted $\triangle d^{max}_m$, can be specified for each media stream m_i. Let D^m_{max} be the maximum size of the data that can be delayed. For example, if the consuming rate of stream m is r^m_c, D^m_{max} is then no larger than $r^m_c \times \triangle d^{max}_m$. Consider the maximum delay given in stream m, Formula (3.1) is then changed to:

$$B_m(t, t^m_{c_s}) = \begin{cases} 0 & \text{if } t < t^m_{l_s}, \\ L_m(t, t^m_{l_s}) - D^m_{max} & \text{if } t^m_{l_s} \le t < t^m_{c_s}, \\ L_m(t, t^m_{l_s}) - D^m_{max} - C_m(t, t^m_{c_s}) & \text{if } t \ge t^m_{c_s}. \end{cases} \tag{3.2}$$

We will now calculate the earliest consumption time of a requested multimedia presentation, given the maximum permitted delay in the steam. Suppose that a solution is to begin display or delivery at time x. That is, $B_m(t, x)$ is at least zero for any time $x \le t \le t^m_{l_f}$, where $t^m_{l_f}$ is the time at which the loading of data stream m is completed. If we compare $B_m(t, x)$ with $B_m(t, t^m_{l_s})$ in the range $x \le t \le t^m_{l_f}$, we see that

$$\begin{aligned} B_m(t, x) &= L_m(t, t^m_{l_s}) - D^m_{max} - C_m(t, x) \\ &= L_m(t, t^m_{l_s}) - C_m(t, t^m_{l_s}) - D^m_{max} + C_m(t, t^m_{l_s}) - C_m(t, x) \\ &= B_m(t, t^m_{l_s}) + C_m(t, t^m_{l_s}) - C_m(t, x). \end{aligned} \tag{3.3}$$

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

$$C_m(t, t_{l_s}^m) - C_m(t, x) \geq k. \tag{3.4}$$

Given D_{max}^m, $B_m(t_{min}, t_{l_s}^m)$ can be determined, where $B_m(t_{min}, t_{l_s}^m)$ has the minimum value at t_{min}^m [GC92a]. Thus, k can be determined in advance. Consequently, x can be determined when the consuming function is given. This start time assumes that the entire stream will be continuously consumed. However, in our context, we assume that the data unit to be accessed is media object rather than the entire stream. Thus, after the consumption or deliver time of stream m is determined, the deliver time of each media object within the stream must also be precisely determined.

3.2.2 Loading and Consuming Times of Media Objects

In order to ensure a hiccup-free presentation based on pre-specified time constraints, 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

$$B_{im}(t, t_{c_s}^{im}) = \begin{cases} 0 & \text{if } t < t_{l_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) & \text{if } t_{l_s}^{im} \leq t < t_{c_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) & \text{if } t \geq t_{c_s}^{im}. \end{cases} \tag{3.5}$$

With the maximum delay, the amount of buffer space that must be allocated for object o_{im} at any given time t is then

$$B_{im}(t, t_{c_s}^{im}) = \begin{cases} 0 & \text{if } t < t_{l_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) - D_{max}^{im} & \text{if } t_{l_s}^{im} \leq t < t_{c_s}^{im}, \\ L_{im}(t, t_{l_s}^{im}) - D_{max}^{im} - C_{im}(t, t_{c_s}^{im}) & \text{if } t \geq t_{c_s}^{im}. \end{cases} \tag{3.6}$$

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:

$$B_{im}(t, t_{c_s}^{im}) \quad = \quad L_{im}(t, t_{l_s}^{im}) - D_{max}^{im} - C_{im}(t, t_{c_s}^{im})$$

$$= L_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{l_s}^{im}) - D_{max}^{im} + C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im})$$
$$= B_{im}(t, t_{l_s}^{im}) + C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}). \quad (3.7)$$

Let k_{im} be determined by $B_{im}(t, t_{l_s}^{im}) \geq -k_{im}(k_{im} \geq 0)$. We then have

$$C_{im}(t, t_{l_s}^{im}) - C_{im}(t, t_{c_s}^{im}) \geq k_{im}. \quad (3.8)$$

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 (3.8).

We consider that the consuming function of each object can be approximated as linear, that is,

$$C_{im}(t, x) = \begin{cases} 0 & \text{if } t < x, \\ r_c^{im}(t - x) & \text{if } t \geq x, \end{cases} \quad (3.9)$$

where r_c^{im} is the consuming rate of object o_{im}. We then have the consumption time of the first object of the stream m based on Formula (3.8), assuming the loading of the stream starts right after it is requested:

$$x \geq t_{l_s}^{im} + \frac{k_{im}}{r_c^{im}}. \quad (3.10)$$

As a result, the consumption times of the rest of the media objects are fixed by the specified time constraints. Using (3.8), the preloading times for these media objects can be defined as

$$t_{l_s}^{im} \leq t_{c_s}^{im} - \frac{k_{im}}{r_c^{im}}. \quad (3.11)$$

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.

We will now determine the latency in the presentation. On one hand, when the loading rate is greater than the consumption rate, the latency in the system is essentially due to delays in the system. On the other hand, when the loading rate is less than the consumption rate, the latency is found to be dependent on the size of the first object as well as the loading and consumption rates. Let the start time of consumption of the first object be x. Following (3.10), the minimum latency is:

$$\triangle t_{lat} = x - t_{l_s}^{im} = \frac{k_{im}}{r_c^{im}}. \quad (3.12)$$

We will now determine k_{im}. We consider that both the loading and consumption rates of the media objects can be approximated as linear. The underlying assumption in the case of the loading function is that the seek time is much less than the load time. Thus, L_{im} can be approximated as a linear function represented as

$$L_{im}(t, t_{l_s}^{im}) = \begin{cases} 0 & \text{if } t < t_{l_s}^{im}, \\ r_l^{im}(t - t_{l_s}^{im}) & \text{if } t \geq t_{l_s}^{im}. \end{cases} \tag{3.13}$$

where r_l^{im} is the loading rate of object o_{im} of stream m.

Given (3.6), (3.9), and (3.13), we know that the minimum value of $B_{im}(t, t_{l_s}^{im})$ occur at time $t_{c_f}^{im}$:

$$B_{im}(t, t_{l_s}^{im}) \quad \geq \quad -[(t_{c_f}^{im} - t_{l_s}^{im})(r_c^{im} - r_l^{im}) + D_{max}^{im}]. \tag{3.14}$$

Thus, k_{im} is $(t_{c_f}^{im} - t_{l_s}^{im})(r_c^{im} - r_l^{im}) + D_{max}^{im}$. Furthermore, $t_{c_f}^{im} = x + \frac{|o_{im}|}{r_c^{im}}$, where $|o_{im}|$ is the size of object o_{im}. Substituting in (3.12), we get the latency in the presentation to be

$$\triangle t_{lat} = x - t_{l_s}^{im} = \frac{|o_{im}|}{r_c^m}\left(\frac{r_c^{im}}{r_l^{im}} - 1\right) + \frac{D_{max}^{im}}{r_l^{im}}. \tag{3.15}$$

In the absence of delays, the latency in the presentation is:

$$\triangle t_{lat} = x - t_{l_s}^{im} = \frac{|o_{im}|}{r_c^{im}}\left(\frac{r_c^{im}}{r_l^{im}} - 1\right). \tag{3.16}$$

Note that the delays at the server can be attributed to varying disk seeking times and loading delays, while network delays and delays arising due to system load are the important sources of delays at the client.

3.3 Buffer Sharing

In this section, we will discuss the least upper bound on the buffer requirement at both client and server sites. In particular, we will present the calculation of times at which objects attain maximum buffer requirement and compare buffer consumption with or without buffer sharing.

3.3.1 Maximum Buffer Consumption

In the analysis given in Section 3, multiple streams are simultaneously supported by assuming that, for each stream m, enough objects of m are loaded into memory to allow smooth consumption of m. Buffers are needed to satisfy this criterion. Clearly, the size of buffer at any point in a time interval $[a, b]$ cannot be pre-determined. This problem may be circumvented by allocating maximum buffer required for each object presented in $[a, b]$. However, this does not give an optimal buffer requirement, leading to a possible decrease in the server utilization and lower the number of streams a client can handle concurrently. To alleviate these problems, we use buffer sharing to reduce maximum buffer requirements.

Clearly, if the loading rate of an object o_{im} is greater than its consumption rate, the object attains its 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 the object attains its maximum buffer requirement at the beginning of consumption phase, at time $t = t_{c_s}^{im}$, given $t_{c_s}^{im} \in [a, b]$. We denote the time at which object o_{im} in stream m 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. Thus, the time at which object o_{im} of stream m attains its maximum buffer requirement is defined as:

- $r_l^{im} > r_c^{im}$:

$$t_{max}^{im} = \begin{cases} t_{l_f}^{im} & \text{if } a \leq t_{l_f}^{im} \leq b, \\ a & \text{if } t_{l_f}^{im} < a, \\ b & \text{if } t_{l_f}^{im} > b. \end{cases} \tag{3.17}$$

- $r_c^{im} > r_l^{im}$:

$$t_{max}^{im} = \begin{cases} t_{c_s}^{im} & \text{if } a \leq t_{c_s}^{im} \leq b, \\ a & \text{if } t_{c_s}^{im} < a, \\ b & \text{if } t_{c_s}^{im} > b. \end{cases} \tag{3.18}$$

Given that each object o_{im} in stream m needs the maximum buffer space at $t = t_{max}^{im}$ in the interval $[a, b]$, the maximum buffer requirement for all objects presented in the interval is simply the sum of all maximum buffers over all the objects presented in interval $[a, b]$. Let B_{tot}^m denote the maximum buffer requirement over all the objects in stream m that are presented in the interval $[a, b]$:

$$B_{tot}^m = \sum_{i=1}^{n_m} B_{im}(t_{max}^{im}, t_{c_s}^{im}), \tag{3.19}$$

where n_m is the total number of objects in stream m in the interval. Therefore, the total buffer requirement over all the streams is

$$\begin{aligned} B_{tot} &= \sum_{j=1}^{n} B_{tot}^j, \\ &= \sum_{j=1}^{n} \sum_{i=1}^{n_j} B_{ij}(t_{max}^{ij}, t_{c_s}^{ij}). \end{aligned} \tag{3.20}$$

where n is the total number of streams and n_j is the total number of objects in stream j in the interval.

We will now discuss the buffer sharing of streams. As defined in (3.20), the total buffer requirement of n media streams is based on the assumption that each stream occupies the sum of maximum buffers required for all its objects presented in each interval.

Figure 3.1 shows the buffer requirement in two intervals $[t_0, t_1]$ and $[t_1, t_2]$ of a presentation involving two streams m_1 and m_2. Assume that the consumption of objects in m_1 can be overlapped (as in images) while delivery of objects in m_2 is sequential (as in audio). Within the interval $[t_0, t_1]$, two objects from stream m_1 (o_{11} and o_{21}) and one object from stream m_2 (o_{12}) are presented, while in the interval $[t_1, t_2]$, one object from each stream is presented (o_{31} and o_{22}). From Figure 3.1, it is clear that stream m_1 does not require $B_{11}(t^{11}_{max}, t^{11}_{c_s}) + B_{21}(t^{21}_{max}, t^{21}_{c_s})$ buffer size at all times in the interval $[t_0, t_1]$, while stream m_2 does not require $B_{21}(t^{21}_{max}, t^{21}_{c_s})$ buffer size at all times in the interval $[t_0, t_1]$. Thus, allowing buffer sharing between the streams would be a simple way to enhance buffer utilization.

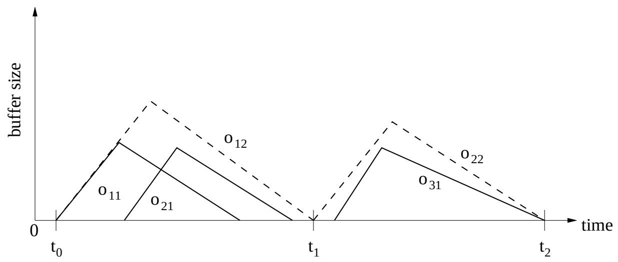

Figure 3.1: Buffer sharing with 2 streams.

A full analysis of buffer sharing would involve finding the maximum buffer requirement applicable to all the objects presented in an interval. This maximum is not necessarily the sum of the maximum buffers over all the objects given by (3.20). We begin by defining a piecewise linear and non-negative function f as

$$f(t) = \begin{cases} 0 & \text{if } t < t_0, \\ c_1(t - t_0) & \text{if } t_0 \le t < t_1, \\ c_2(t - t_2) & \text{if } t_1 \le t \le t_2, \\ 0 & \text{if } t > t_2 \end{cases} \tag{3.21}$$

where c_1 is a positive constant and c_2 is a negative constant. Note that the definition of $f(t)$ is similar to the definition of the buffer function B_{im} in (3.1) with $t_0 = t^{im}_{l_s}$, $t_1 = t^{im}_{l_f} = t^{im}_{c_s}$ and $t_2 = t^{im}_{c_f}$. The following lemma is useful in determining the maximum buffer requirement for all objects in each interval.

Lemma 1 *Let $f_1(t)$, $f_2(t)$, ..., $f_n(t)$ be n functions of the form defined in (3.21), $a \le t \le b$. Assume that $f_1(t)$, $f_2(t)$, ..., $f_n(t)$ attain their maximum at $t^1_{\max}$, $t^2_{\max}$, ..., $t^n_{\max}$ respectively. Let*

$$g(t) = f_1(t) + f_2(t) + \cdots + f_n(t).$$

Then, $\max(g(t)) = \max\{g(t^i_{\max}) \mid 1 \le i \le n, a \le t \le b\}$

Proof. Let t_0 be a point in $[a, b]$ where no function $f_i(t)$ has a maximum (see Figure 3.2). Let S denote the sum of the slopes of functions that have a non-zero value at t_0, that is, $S = \sum_{f_i(t_0)\neq 0} \frac{df_i}{dt}\Big|_{t=t_0}$. If $S \geq 0$, then there exists at least one function $f_i(t)$ whose slope is positive at t_0. Let t_1 be the first point to the *right* of t_0 at which one of the functions with positive slope or one of those functions that were zero at t_0 attains its maximum. We have

$$
\begin{aligned}
g(t_1) - g(t_0) &= \sum_i f_i(t_1) - \sum_i f_i(t_0) \\
&= \sum_i f_i(t_1) - f_i(t_0)
\end{aligned}
$$

Using $\frac{dh}{dx}\Big|_{x=x_i} = \frac{h(x_1)-h(x_2)}{x_1-x_2}$ for a linear function $h(x)$ and for any $x_i \in [x_1, x_2]$, we have

$$
\begin{aligned}
\frac{g(t_1) - g(t_0)}{t_1 - t_0} &= \sum_i \frac{f_i(t_1) - f_i(t_0)}{t_1 - t_0} \\
&= \sum_{f_j(t_0)\neq 0} \frac{df_j}{dt}\Big|_{t=t_0} + \sum_{f_k(t_0)=0} \frac{f_k(t_1) - f_k(t_0)}{t_1 - t_0}
\end{aligned}
$$

The first term equals $S(\geq 0)$ and by choice of t_1 the second term is positive. Therefore,

$$g(t_1) - g(t_0) \geq 0$$

,

$$g(t_1) \geq g(t_0)$$

.

If $S < 0$, by the same argument, it can be shown that $g(t_1) \geq g(t_0)$ for the first point t_1 to the *left* of t_0 at which one of the functions with a negative slope at t_0 or one of those functions that were zero at t_0 attains its maximum. □

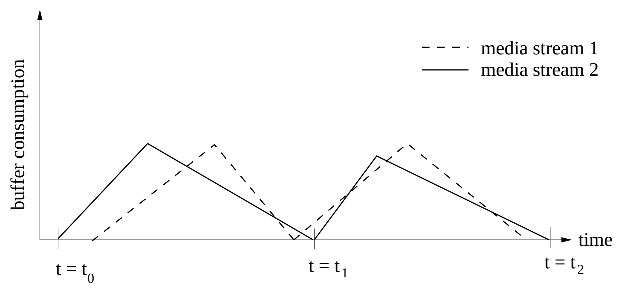

Figure 3.2: Finding the maximum value of a set of functions.

Theorem 1 *The maximum buffer requirement over all the objects in interval [a, b] occurs at a time when at least one of the objects attains its maximum buffer requirement in [a, b].*

Proof. Let $o_{kl}, (1 \leq l \leq n, 1 \leq k \leq n_l)$, be all objects presented in $[a,b]$. Clearly, the buffer consumption functions B_{kl} for all these objects follow the form defined in (3.21). Thus, following Lemma 1, the maximum buffer requirement in $[a,b]$ occurs at one of $t_{kl}^{max}, (1 \leq l \leq n, 1 \leq k \leq n_l)$, and is

$$B_{shar} = \max\{\sum_{j=1}^{n}\sum_{i=1}^{n_j} B_{ij}(t_{max}^{kl}, t_{c_s}^{ij}) \mid 1 \leq l \leq n,\ 1 \leq k \leq n_l\}. \tag{3.22}$$

□

Theorem 2 *Let the total buffer space required in the interval $[a,b]$ without buffer sharing be B_{tot} given by (3.20). Then, $B_{shar} \leq B_{tot}$. Moreover, buffer size B_{shar} is necessary.*

Proof. Let the maximum in (3.22) occur at $l = q, (1 \leq q \leq n)$ and $k = p, (1 \leq p \leq n_q)$. Obviously, the buffer space required at $t = t_{max}^{pq}$ cannot be less than B_{shar}. Hence, B_{shar} is necessary. We have

$$B_{shar} = \sum_{j=1}^{n}\sum_{i=1}^{n_j} B_{ij}(t_{max}^{pq}, t_{c_s}^{ij}).$$

Consider the objects in stream m. Let us denote the total buffer requirement for stream m as B_{shar}^m. We have,

$$\begin{aligned} B_{shar}^m &= B_{1m}(t_{max}^{pq}, t_{c_s}^{1m}) + B_{2m}(t_{max}^{pq}, t_{c_s}^{2m}) + \cdots + B_{n_m m}(t_{max}^{pq}, t_{c_s}^{n_m m}), \\ &\leq B_{1m}(t_{max}^{1m}, t_{c_s}^{1m}) + B_{2m}(t_{max}^{2m}, t_{c_s}^{2m}) + \cdots + B_{n_m m}(t_{max}^{n_m m}, t_{c_s}^{n_m m}), \\ &= B_{tot}^m \end{aligned}$$

Similarly, we can show that

$$B_{shar}^i \leq B_{tot}^i,\ 1 \leq i \leq n$$

$$\begin{aligned} \sum_{i=1}^{n} B_{shar}^i &\leq \sum_{i=1}^{n} B_{tot}^i, \\ B_{shar} &\leq B_{tot}. \end{aligned}$$

□

Example 3 *Consider a presentation involving two media streams m_1 and m_2. Let the buffer consumption profile for both streams be as shown in Figure 3.1. Table 3.1 lists the specifications for both streams. Since the loading rate is greater than the consumption rate for both streams, $t_{l_s} = t_{c_s}$ for all objects and the maximum buffer requirement for each object occurs at the end of the loading phase; that is , at $t = t_{l_f}^{im}$ for object o_{im} of stream m. By (3.20), the maximum buffer requirement*

Streams	m_1			m_2	
Objects	o_{11}	o_{21}	o_{31}	o_{12}	o_{22}
Loading rate, r_l (kB/s)	1000	1000	1000	1000	1000
Consumption rate, r_c (kB/s)	240	240	240	120	120
Loading start, t_{l_s} (s)	0.2	1.2	4.8	0.2	4.5
Loading end, t_{l_f} (s)	1.2	2.8	6.0	2.0	6.4
Consumption start, t_{c_s} (s)	0.2	1.2	4.8	0.2	4.5

Table 3.1: Loading and consumption parameters

for stream m_1 in interval $[t_0, t_1]$ is $B_{11} + B_{21}$ and is equal to $(760 \times 1.0 + 760 \times 1.6) = 1976$ kB. The maximum buffer requirement for stream m_2 in $[t_0, t_1]$ is $B_{12} = 880 \times 1.8 = 1584$ kB. The total buffer requirement is 3560 kB. Similarly, in $[t_1, t_2]$, m_1 requires 912 kB while m_2 requires 1672 kB and the total buffer requirement is 2584 kB.

In contrast, by (3.22), the maximum buffer requirement with buffer sharing for streams m_1 and m_2 in $[t_0, t_1]$ occurs precisely at one of $t = \{1.2s, 2.0s, 2.8s\}$. The total buffer requirement for objects o_{11}, o_{21}, and o_{12} in $[t_0, t_1]$ at $t = 1.2s$ is $(760 + 880 + 0) = 1640$ kB. Similarly, at $t = 2.0s$, the total buffer requirement is $(1584 + 608 + 368) = 2560$ kB, while at $t = 2.8s$, the total buffer requirement is $(176 + 1488 + 1216) = 2880$ kB. The maximum buffer requirement in the interval is $\max\{1640, 2560, 2880\} = 2880$ kB. In $[t_1, t_2]$, the maximum buffer requirement with buffer sharing is be found to be 2232 kB. Thus, buffering sharing reduced buffer consumption by 19% in $[t_0, t_1]$ and by 14% in $[t_1, t_2]$.

3.3.2 Buffer Consumption with Synchronization

We will now apply the basic principles of buffer consumption discussed above to synchronized presentation of multiple media streams in the event of delays.

We define a *synchronization point* to be a point held in common by all participating media objects in synchronized streams. Thus, synchronization points specify nodes in media streams at which the synchrony of presentation must be checked and maintained. In general, the temporal relationship between two atomic objects from different media streams may conform to any of the thirteen temporal relationships described in [All83], which is demonstrated in Figure 3.3. We specifically introduce three types of synchronization points among media objects; these are *intra*-synchronization point, *inter*-synchronization point, and *middle*-synchronization point. Intra-synchronization points specify sequential time-constraints among two media objects. Inter-synchronization points specify exact parallel start and end relationships among objects. Middle-synchronization points specify

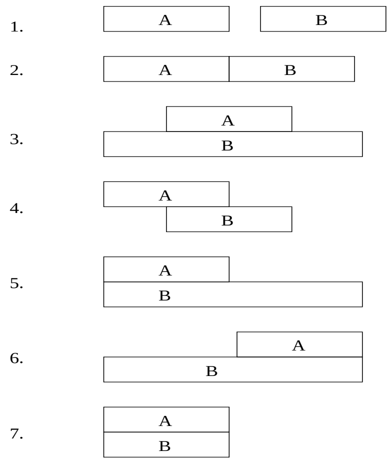

Figure 3.3: Typical temporal relationships between objects.

the situations that one object must start in the middle of another object. For example, in Figure 3.3, cases 1 and 2 have intra-synchronization points between objects A and B; cases 5, 6, and 7 have inter-synchronization points between objects A and B; and cases 3, 4, 5, and 6 have middle-synchronization points between objects A and B. The detailed discussion on synchronization can be found in [ZG95a].

All objects that exist at an inter-synchronization point must either start or finish consumption at that point, that is, $t_{c_s}^{jm} = t_{c_s}^{kn}$ or $t_{c_f}^{jm} = t_{c_f}^{kn}$ for all objects o_{jm} and o_{kn}, $m \neq n$, that exist at the synchronization point. These constraints could lead to one of the following presentation in an interval $[t_i, t_{i+1}]$:

1. All objects begin their consumption at the same time t_i but complete consumption at different times. In this instance, the inter-synchronization point is set at $t_s = t_i$ in the interval $[t_i, t_{i+1}]$.

2. All objects are constrained to finish consumption at the same time but do not start consumption concurrently. In such a case, the synchronization time t_s would equal the upper bound of the interval t_{i+1} and there would be no constraint on the lower bound.

3. All objects start consumption at t_i and end consumption at t_{i+1}. In this instance, two inter-synchronization points are fixed in the interval $[t_i, t_{i+1}]$, one at t_i and the other at t_{i+1}.

The above three cases can occur under one of the following conditions: (a) $r_c^m > r_l^m, \forall m(1 \leq m \leq n)$; (b) $r_l^m > r_c^m, \forall m(1 \leq m \leq n)$; and (c) $r_c^m > r_l^m, \exists m(1 \leq m \leq n)$. Figure 3.4 shows the above three cases under condition (a). Under condition (a), $t_{l_s}^m < t_{c_s}^m, \forall m(1 \leq m \leq n)$. Therefore, the maximum buffer requirement occurs at time $t = t_{c_s}^m$. Condition (b) implies $t_{l_s}^m = t_{c_s}^m, \forall m(1 \leq m \leq n)$ provided there are no delays in the presentation of objects. Under such a condition, the maximum buffer requirement would occur at $t = t_{l_f}^m$, $t_i \leq t \leq t_{i+1}$. Under condition (c), since all the objects do not start their consumption at their maximum or minimum buffer requirement, the maximum could occur at any point inside the interval. If all objects in the interval $[t_i, t_{i+1}]$ are participating at a inter-synchronization point, we have the following result:

Corollary 1 *Given that the bounds of the interval* $[t_i, t_{i+1}]$ *are inter-synchronization points involving all objects presented in the interval,*

1. *If* $r_c^m > r_l^m$, $\forall m$, $1 \leq m \leq n$, *the maximum buffer requirement in the interval occurs at one of the bounds and is equal to* $B_{inter} = \sum_j \sum_i B_{ij}(t_{max}^{ij}, t_{c_s}^{ij})$. *In contrast, if all streams in the interval do not participate at the inter-synchronization points, the maximum buffer requirement in* $[t_i, t_{i+1}]$ *is* B_{shar}.

2. *If* $r_c^m < r_l^m$, $\forall m$, $1 \leq m \leq n$, *the maximum buffer requirement in the interval occurs at one of the points inside the interval and is equal to* B_{shar}.

3. *If* $r_c^m < r_l^m$, $\exists m$, $1 \leq m \leq n$, *then the maximum buffer requirement in the interval occurs at one of the points inside the interval and is equal to* B_{shar}.

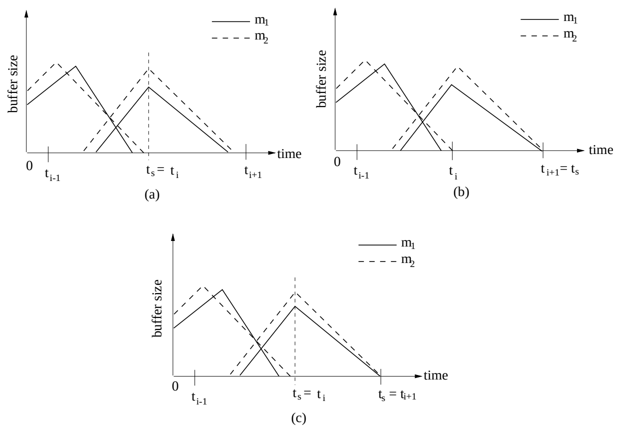

Figure 3.4: Buffer consumption in the presence of inter-stream synchronization. (a) concurrent start of consumption; (b) concurrent end of consumption; and (c) concurrent start and finish of consumption.

Two types of intra-synchronization points occur within a stream. In the first type, the consumption of an object o_{im} begins immediately after the preceding object o_{i-1m} is consumed. In other words, $t_{c_f}^{i-1m}$ of object o_{i-1m} equals $t_{c_s}^{im}$ of object o_{im}. With intra-synchronization points of the second type, the consumption of an object o_{im} follows after a time interval $\triangle t$ after the preceding object o_{i-1m} is consumed. As before, we have the following conditions: (a) $r_c^m > r_l^m$ for stream m; and $r_c^m < r_l^m$ for stream m. Figure 3.5 illustrates intra-synchronization under condition (a). Under condition (a), the maximum buffer requirement occurs at the one of the bounds of the interval while under condition (b) the maximum occurs at a point inside the interval and is given by the following result:

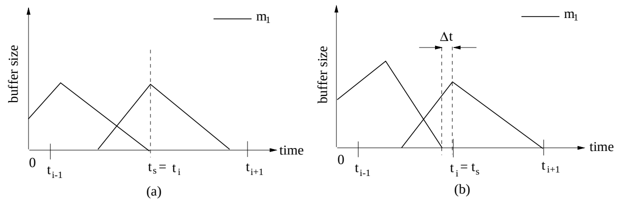

Figure 3.5: Buffer consumption in the presence of intra-stream synchronization. (a) concurrent start and finish of consumption of two consecutive objects; (b) delay between the start and finish of consumption of two consecutive objects.

Corollary 2 *Given that the bounds of the interval $[t_i, t_{i+1}]$ are intra-synchronization points involving all objects $\in$ stream m that are presented in the interval,*

1. *If $r_c^m > r_l^m$ the maximum buffer requirement over all the objects occurs at one of the bounds of the interval and is equal to $B_{intra} = \max\{B_{im}(t_{max}^{jm}, t_i), B_{im}(t_{max}^{km}, t_{i+1}\}$ for some objects o_{jm} and o_{km} participating at the intra-synchronization points t_i and t_{i+1}, respectively.*

2. *If $r_c^m < r_l^m$ the maximum buffer requirement occurs at a point inside the interval excluding the bounds and is equal to B_{shar}.*

Middle-synchronization points can involve more than one media stream. A media stream m with objects $o_{im}, (1 \leq i \leq n_m)$ has a middle-synchronization point if there exists another media stream l with objects $o_{jl}, (1 \leq j \leq n_l)$ such that $t_{l_s}^{km} < t_{l_s}^{pl} < t_{l_f}^{km}$ or $t_{l_s}^{pl} < t_{l_s}^{km} < t_{l_f}^{pl}$ for some objects $o_{km} \in m$ and $o_{pl} \in l$. In Figure 3.6, t_s is a middle-synchronization point where the second object of stream m_1 starts consumption after the first object of stream m_1 starts consumption but before the first object finishes consumption. Note that Figure 3.6 illustrates middle-synchronization under the condition $r_l^m > r_c^m, \forall m (1 \leq m \leq n)$.

Corollary 3 *Given that the bounds of the interval $[t_i, t_{i+1}]$ are middle-synchronization points, the maximum buffer requirement for the interval is B_{shar} for both conditions, $r_c^m > r_l^m, \forall m (1 \leq m \leq n)$ and $r_c^m < r_l^m, \forall m (1 \leq m \leq n)$.*

3.4 Buffer Prefetching and Replacement

In this section, we will discuss buffer prefetching and replacement based on the principles discussed above. The proposed prefetching and replacement strategy satisfies the requirement of continuity

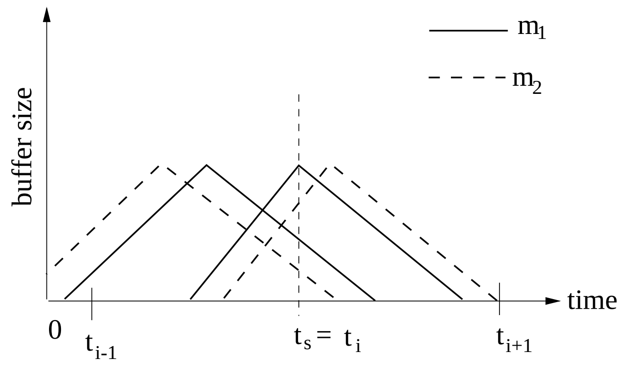

Figure 3.6: Buffer consumption in the presence of middle-synchronization.

and synchrony in the presentation of multimedia objects, while minimizing the number of buffer faults.

In order to provide support for buffering based on the principles discussed in Sections 3 and 4, a multimedia database system must provide efficient buffer *allocation* and *replacement* strategies. One of the aims of an allocation strategy is to reduce the number of *buffer faults* and thereby reduce delays in the presentation. A buffer fault occurs when the object needed for display is not in the buffer. The main goal of a replacement strategy is to replace those buffers that are least unlikely to be referenced during a certain time interval. Certainly, an easy way to circumvent the problem of buffer faults is to always allocate sufficient buffer space, but this could lead to very low buffer utilization. On the other hand, reducing the buffer space could lead to a large number of buffer faults.

The time constraints specified in the presentation define the consumption times of the objects contained in the presentation. A presentation is characterized by a list of tuples $\langle o_{im}, t_{c_s}^{im}, \triangle t_{im}\rangle$ that represent the objects over all streams in the presentation. Each media object o_{im} is associated with a START event, corresponding to $t_{c_s}^{im}$, and an END event corresponding to $t_{c_f}^{im} = t_{c_s}^{im} + \triangle t_{im}$. All START and END events are classified into layered GROUPs based on the time constraints pertaining to the events. Moreover, all START events in GROUP$_i$ $(1 \leq i \leq n)$ are invoked simultaneously; all END events in GROUP$_i$ $(1 \leq i \leq n)$ are invoked simultaneously; all START events in GROUP$_i$ $(1 \leq i \leq n)$ can be invoked only after all END events in the same GROUP have terminated; and events in GROUP$_i$ always have a higher priority over events in GROUP$_{i+1}$. Thus, each GROUP includes all events occurring at the same time in the presentation and corresponds to a synchronization point in the presentation. Enforcing synchronous delivery of objects at every synchronization point will generate a *synchronous* presentation [ZG95a].

The presentation is segmented into time intervals corresponding to the synchronization points. Thus, interval $[t_0, t_1]$ is constructed from the first two synchronization points p_0 and p_1, occurring at $t = t_0$ and $t = t_1$, respectively. The lower bound of the first interval $[t_0, t_1]$ is determined dynamically by calculating the actual consumption time of the starting objects in the presentation using (3.10), assuming these objects are loaded as soon as the presentation is requested. The upper bound of $[t_0, t_1]$ and the bounds of the rest of the intervals are then determined. The consumption and retrieval of objects are interleaved to reduce buffer requirements and latency of the presentation. Depending on the maximum delay as well as the loading and consumption rates of stream m, an object o_{im}, that will be consumed in interval $[t_l, t_{l+1}]$, has to be prefetched in an earlier interval or the present interval to ensure continuous presentation of the object. To minimize buffer requirements, all objects are not prefetched at the same time. After the consumption times of the starting objects are determined, the consumption times of the rest objects in the presentation are then determined. The prefetching times of such object $\langle o_{im}, t_{c_s}^{im}, \triangle t_{im} \rangle$ can be determined using (3.11).

For each object o_{im}, prefetching at $t_{l_s}^{im}$ is necessary and sufficient to ensure its *continuous* presentation beginning at $t_{c_s}^{im}$. The prefetching times can be sorted and placed in the appropriate intervals. Thus, each interval has a list of objects whose prefetching times belong to the interval. In each interval, objects in this list are prefetched at the appropriate times to ensure smooth presentation of the objects in a later interval. Dealing with a finite buffer size in each interval necessitates a replacement strategy to minimize buffer faults and ensure a continuous presentation. We use the Least Recently consumed (LRC) strategy that replaces the object with the smallest t_{c_f} in the buffer with the new object. In other words, the object that finished consumption the earliest is removed first from the buffer. Using an LRC replacement strategy helps in dealing with delays more gracefully. In the event of delays, the last presented object in the faster stream is most likely to be presented over and over again (object duplication) to correct the skew in the presentation. Thus, an LRC strategy guarantees that the most recently consumed objects are always in the buffer.

Algorithm 3 summarizes the above scheme. This scheme ensures that the buffer requirement in each interval $[t_i, t_{i+1}]$ is always $\leq B_{shar}$ given in (3.22). By Theorem 2, B_{shar} size of buffer is necessary and sufficient in each time interval $[t_i, t_{i+1}]$. We have thus proven:

Theorem 3 *Let B^i denote the total buffer size over all the objects presented in interval $[t_i, t_{i+1}]$. Algorithm 3 minimizes the total buffer requirement by minimizing $\max_{0 \leq i \leq n} B^i$ where n is the total number of intervals in the presentation and ensures both continuity and synchronization in the presentation.*

We have shown a buffer prefetching algorithm that maintains synchronization constraints thereby minimizing skew and jitter [LG90a] in a presentation while minimizing the buffer requirements. Furthermore, the algorithm handle delays in the presentation more gracefully by using an LRC replacement strategy for buffer replacement. This allows us to minimize the buffer faults and hence average delays in the presentation, thereby decreasing the additional buffer size required to store the delayed objects and make the presentation continuous.

Algorithm 3 (buffer prefetching and replacement)

Input: the presentation; B_{max}, the maximum buffer.

Segment the presentation into n time intervals coinciding with
the synchronization points in the presentation.
Calculate the starting load times(t_{l_s}) for all objects in the
presentation and place them in the appropriate time interval.

buffer_size $\leftarrow 0$
BUFFER $\leftarrow \{\}$
clock $\leftarrow t_0$
for $i = 0$ **to** $n - 1$ **do**
    if $B_{shar} > B_{max}$ **then return**
    $S \leftarrow \{o_{kj} \mid t_i \leq t_{l_s}^{kj} \leq t_{i+1}$ or $t_i \leq t_{l_f}^{kj} \leq t_{i+1}\}$
        // S is sorted based on time
    foreach $obj \in S$ **do**
        while clock $< t_{l_s}^{obj}$ **do**
            increment clock
        if buffer_size+$|obj| \leq B_{shar}$
        then BUFFER $\leftarrow$ BUFFER $\cup\{obj\}$
            buffer_size $\leftarrow$ buffer_size+$|obj|$
        else replace object obj' with smallest t_{c_f} in BUFFER with obj
            buffer_size $\leftarrow$ buffer_size $+|obj| - |obj'|$
            BUFFER $\leftarrow$ (BUFFER$-\{obj'\}) \cup \{obj\}$
return

Chapter 4

QoS Management in Multimedia Systems

Along with synchronization and buffer management, there are also important human interaction factors which are particular to multimedia applications. These factors define specific requirements that must be considered in building the multimedia tools. In particular, the tools must be able to handle various types of delays and to support user interactions such as skip, pause, and fast forward/backward in a manner that is acceptable to on-line users. To effectively and efficiently support these functions, a flexibly adjustable quality of service (QoS) [LG90a] must be supported. Since these specific needs may be different from those of other multimedia applications, tools that integrate these particular QoS parameters must be developed.

4.1 Quality-of-service Requirements and Correctness Criteria

In this section, we will first discuss QoS parameters and the effect of these parameters in the scheduling of multimedia presentations. We will then define correctness criteria for the executions of multimedia tasks. As illustrated below, QoS parameters must be considered in the definitions of these correctness criteria. The reader is referred to [ZG95b] for a detailed discussion on QoS management in multimedia systems.

The scheduling of multimedia tasks includes the scheduling of time-dependent delivery operations, synchronized delivery enforcement among multiple media streams in a multimedia task, concurrent execution of multimedia tasks, and delivery delay recovery. A correctness criterion in this context must verify that delivery operations are performed according to a predefined synchronization pace and within the time constraints imposed on tasks. Since the correctness of time-based presentations depends on the accuracy of timing that must be maintained on media streams, the execution result of a multimedia task is a question of quality rather than consistency. We must thus formulate new correctness criteria for the executions of multimedia tasks which define acceptable quality in real-time. Several important QoS parameters must be considered in these correctness criteria.

Little and Ghafoor [LG90a] 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 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 maintain the parameters of speed ratio and utilization to be 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 then, in case that an audio object is delayed, drop some video frames in the corresponding video object to maintain synchronization between the two objects. If the scheduler tries to maintain 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.

Multimedia presentations in the domain of education or training are normally instruction-oriented. Such presentations should usually last two to three hours. The media streams involved in a single presentation may include a combination of audio and slides, or a combination of audio and video as well as text. Preserving synchronization requirements on such presentations must be the top priority in defining QoS services. Distortion between synchronized media objects may result in misunderstanding of the teaching materials. Figure 4.1 demonstrates an experimental example of the presentation of audio and video streams. Table 4.1 provides the measurement of average delay, jitter, and skew. In this experiment, synchronization constraints are implicitly imposed by the specification of time constraints on media objects. As the synchronization points are not explicitly enforced in the presentation, delays occurred in the individual objects resulted in mismatches among the displayed media objects from different streams. High values in jitter and skew are reported in the table. Such mismatches may cause confusion in understanding the materials.

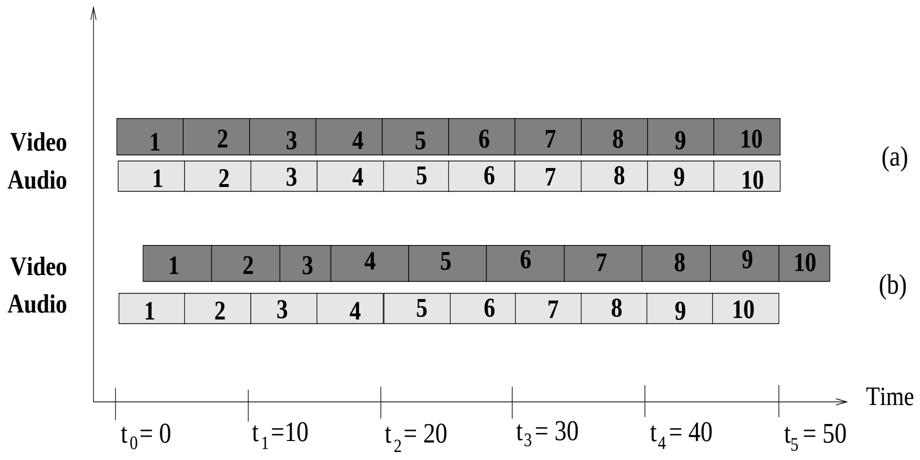

Figure 4.1: Presentation of audio and video streams : (a) nominal, (b) actual presentation

`Parameter`	$t_0 - t_1$	t_1	$t_1 - t_2$	t_2	$t_2 - t_3$	t_3	$t_3 - t_4$	t_4	$t_4 - t_5$	t_5
Average Delay	1.0	-	0.75	-	1.75	-	2.50	-	2.50	-
Skew	2.0	-	1.5	-	2.5	-	4.5	-	5.0	-
Jitter	-	2.0	-	2.0	-	4.0	-	5.0	-	5.0

Table 4.1: Parameter values for presentation without synchronization

Thus, our primary goal on formulating acceptable QoS parameters should be to avoid any deviations from the synchronization constraints associated with the media streams. 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. Upon these measures, we require that the presentation tools will minimize the possibility of jitter and skew.

We will now define a correctness criterion for the execution of a multimedia task and then identify those schedules to be considered to be correct. As with conventional database query executions, the semantics of a multimedia task determine the correctness of its execution. Unlike conventional database query executions, however, the time constraints defined within multimedia tasks assume a position of prime importance. We thus introduce the following semantic correctness criterion:

Definition 7 (Correct execution) *The execution of a multimedia task T is correct if the time constraints specified within T are preserved.*

This semantic correctness criterion is theoretically applicable to the executions of multimedia tasks. However, in a practical, delay-prone system, this criterion cannot be applied directly by the scheduler to enforce the execution of multimedia tasks. Given the pervasive nature of delays, a strict application of this rule would result in the aborting of the vast majority of multimedia tasks. A more realistic scheduling criterion is therefore needed. We introduce the concept of *acceptable* schedules by incorporating the effect of delays into the definition of schedules.

Definition 8 (Acceptable execution) *The execution of a multimedia task is acceptable if and only if all delays occurring at the defined synchronization points are within the permissible QoS ranges.*

Note that the scheduler at a client site need not consider the concurrent execution of multiple multimedia tasks. It must control the presentation of multiple media streams to a single user. There is need to have a central scheduler to manage the real-time executions of all multimedia tasks at the server site.

4.2 Incorporation of Adaptability - A Best-Effort Approach

We will now discuss the incorporation of adaptability into system design on both the client and server sides. Adaptability is realized throughout the system via a best-effort approach. That is, the system will dynamically adjust its state to best service client requests as specified in the QoS requirements. The implementation of such dynamic adjustments will be discussed below.

On the client side, the system must maintain the synchronized presentation of multiple media streams. In order to maintain hiccup-free presentations in the event of network delays, a local cache must be established to buffer some display data. The system may also need to dynamically skip or drop some data to maintain the synchronization of different streams. Two additional difficulties must be addressed on the local side. First, the process of decoding the media objects in a stream may sometimes lag, resulting in a reduced display rate. As a result, the local cache may become inundated. In this situation, the system should be able to request a lower transmission rate from the network. Second, due to network delays, the local cache may sometimes experience a shortage of display data. In this situation, the system should be able to request a higher transmission rate from the network.

On the server side, the system must control client admissions, schedule client requests, and manage a shared buffer for all admitted clients. The scheduling of client requests should exploit the periodic nature of media data playback by processing client requests in *intervals* of time (or rounds, as defined in [HGP94]). Within each interval, all currently admitted clients are serviced in rounds, so that the retrieval of a portion of the data for each client will not result in jitter. Following the best-effort approach, the service time allocation of the client rounds in each interval will be based on current client demands. Thus, the rounds may not be evenly divided in each interval, and the round sizes for each client in each interval may vary.

Buffer management functions within the server must determine the upper bound on buffer consumption for all active clients. As the retrieval rate and service time for each client request may vary in different intervals, the calculation of the upper bounds on buffer consumption must be interval-based. Buffer consumption becomes an important factor that must be considered in admitting new clients. In addition, a data allocation and replacement strategy must be designed to optimize the utilization of retrieved data. Issues of buffer allocation and replacement are addressed in Chapter 3.

We will now discuss the application of the best-effort approach to an adaptable admission control strategy. This strategy must consider both service time and buffer size. A new client must not be admitted to a given interval unless that client's requests can be guaranteed within a given number of consecutive intervals. Sufficient service time must be allocated in each interval to retrieve an adequate number of data blocks for the client and to guarantee that no jitter will occur. Any extra buffer space available can be used to reduce the number of data blocks that must be allocated with a given interval by permitting the read-ahead of data blocks prior to the interval. For example, assume that the server can retrieve 10 data blocks from disk in each interval. Consider that the next five consecutive intervals $I_{l_1}, I_{l_2}, I_{l_3}, I_{l_4}, I_{l_5}$ are already allocated to services of 5, 6, 4, 7, 9 blocks, respectively, requested by current clients. Assume that two new clients R_1 and R_2 each need a minimum of two blocks retrieved from disk in each interval. If these data requests are distributed uniformly over the five intervals as requested, neither of these two new clients can be admitted, since 9 blocks are already occupied within interval I_{l_5}. However, if the available buffer space permits read-ahead, allocating 3, 2, 4, and 1 blocks to R_1 over $I_{l_1}, I_{l_2}, I_{l_3}, I_{l_4}$, respectively, and 2, 4, 2, 1, and 1 to R_2 over $I_{l_1}, I_{l_2}, I_{l_3}, I_{l_4}, I_{l_5}$, respectively, will permit both clients to be admitted. This arrangement is preferable to allocating 5 blocks to either R_1 or R_2 in I_{l_1}; the latter arrangement would prohibit the admission of one of the new clients, and the denied client would experience latency.

We thus formulate the following two rules to guide the design of an adaptable admission control strategy:

- **Minimum-fit-in**: new clients must not be admitted unless their minimum data requests in all involved future intervals can be guaranteed.

- **Queuing latency**: clients should be admitted in a fashion which results in minimum latency.

The incorporation of adaptability on the server side of the system, including client scheduling, buffer management, and admission control, will be discussed in detail in the following Chapter.

Chapter 5

Multimedia Server Functionalities

Multimedia servers play a central role in the delivery of continuous media in distributed systems. The servers 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 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 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 [GC92a, LD93, HGP94].

5.1 Previous Work

The low-level design of multimedia servers has become the focus of several research projects in the recent past [RV91, GC92b, LD93, VR93]. Neufeld and co-workers [GDN95] propose a scheme that employs a scheduling policy based on rounds. A new client request is admitted if the additional number of blocks that are read do not exceed the maximum blocks that can be read in each round. This strategy does not consider the fact that each client request can have different QoS parameters. Moreover, delays due to network and bounds on buffer sizes are not addressed by their strategy. Vin and co-workers [HGP94] propose admission control algorithms to determine whether a client should be admitted for service by a server. However, they do not consider buffer management while calculating the admission control criteria. In this work, we propose a simple admission control policy that incorporates both QoS of each client request as well as the buffer size at the server site.

We begin with outlining a scheduling strategy for servicing multiple client requests and then illustrate buffer consumption profiles at the server based on the buffer model described in Chapter 3. An admission control strategy based on these strategies is then proposed.

5.2 Scheduling Multiple Clients

Consider a distributed multimedia server simultaneously servicing n client requests (for example, $R_1, R_2, \ldots R_n$, respectively). 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 processing time at the server is divided into intervals of fixed size. Each interval is then dynamically divided into n *rounds*. The server processes these n current requests via periodic rounds [HGP94], retrieving a fraction of the segments for each client request in each round. Thus, interval I_k consists of $round_{ik},\ i = 1, \ldots, n$. 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 would then be proportional to the number of segments read from the disk and the sum of the durations of all the rounds would be

$$\Sigma_i^n \tau_{ik} = T,$$

where τ_{ik} is the duration of round i in interval I_k. The number of segments to be retrieved for

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

request R_i in $round_{ik}$ would be

$$N_{ik} = r_{c_i} T.$$

Note that, as the number of clients serviced increases in each interval, the duration of each round decreases. Thus, increasing the number of client requests would reduce the QoS at each client, as the data each client can access in each round diminishes successively.

Due to differences in media data compression techniques and in the relative placement of segments on the disk, the actual time τ_{ik} taken to retrieve N_{ik} segments in $round_{ik}$ could be different in any two rounds. Overflow and underflow rounds [HGP94] occur if $\tau_{ik}^{act} > \tau_{ik}$ and $\tau_{ik}^{act} < \tau_{ik}$, respectively. In an underflow round, the server can read additional segments into the buffer up to a maximum round duration. This capability, termed *read-ahead*, helps to reduce delays in the retrieval of objects in future rounds. In contrast, in the event of an overflow round, the server delays the retrieval of the remaining segments until the next round (i.e., $round_{i+1k}$). This delay need not necessarily reduce the QoS parameters for each client when data from 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 $\lambda_{ik} = N_{ik}^{act}/\tau_{ik}$, the maximum seek time for a segment belonging to a request R_i can be calculated by

$$\lambda_i^{max} = \max\{\lambda_{ij} \mid 1 \leq j \leq n_r\}, \tag{5.1}$$

where n_r is the number of rounds completed to date.

5.3 Buffer Management

Traditional buffer management strategies have addressed the idea of a *hot set* of data to retain in the buffer [AD90]. The LRD strategy [GZ96a] uses the least recently displayed object to be replaced by the new object in case of a buffer miss (similar to a page miss). This is based on the following observation. For any n consecutive requests for the same object, the streams following the first request can read the data read into the buffer by the first request, until of course, the data is retained in the buffer until all the $n-1$ streams read the data. This makes it possible to maximize the number of client requests that can be serviced with the data that is already stored in the buffer thereby decreasing buffer miss probability and loading delays.

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 $B_{ik}(t, t_{c_s}^{ik})$. 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 5.1 shows the buffer consumption profile for two simultaneous requests. In round 0, the server loads in the data for client 0, while in round 1 data for client 1 is loaded. Note that in round 1, whatever is left of the data read for client 0 in round 0 is also transmitted by the communication manager. Using the buffer consumption model defined in Section 2, 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 5.1 occurs at t_0 and occurs at t_1 in the interval 1.

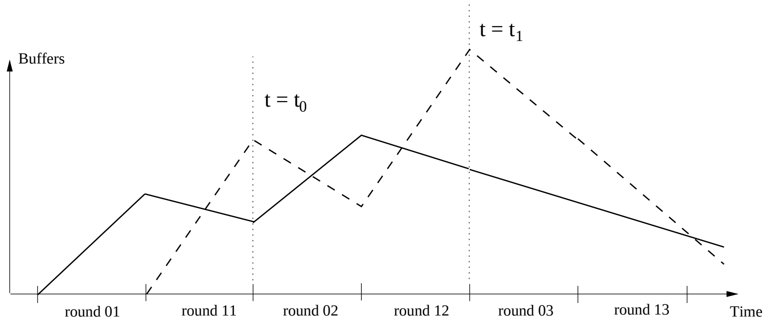

Figure 5.1: Buffer consumption for two client requests

5.4 Admission Control

Guaranteeing continuous playback for each client request requires that the retrieval time of data in a given round will not exceed the playback time of the data retrieved in that round. To achieve this end, the server must control the admission of new client requests, determining whether they can be accommodated without violating the deadline requirements of the current client requests. The formulation of an admission control strategy depends upon the QoS requirements of each client and the shared resources at the server site.

Different clients will have varying QoS parameters which should be exploited by the multimedia server in processing 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, α sets the fraction of data which can be dropped or the permission time through which the server can delay the transmission of segments.

Another important parameter in admission control is the buffer limitation. Servicing a new request would entail an increase in the buffer requirements. Admitting a new request should not result in the violation of the deadline requirements of requests being currently serviced. A simple strategy would be to ensure that the total buffer consumption throughout the next m successive intervals will not exceed the maximum buffer size B_{max}. To do this, an upper bound must be placed on the seek times for each request. These values for current requests can be found using the analysis given in Section 4.1. Upper bound values for new requests (for example, λ_{max}) can be determined 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. Adding a new request to the current set of requests would decrease the duration of each round in an interval. Let N_{ik}^n denote the number of blocks of request R_i actually read in $round_{ik}$ when there are n requests being currently serviced. Clearly, $N_{ik}^1 = \frac{T}{\lambda_i^{max}}$. The number of segments read for successive addition of requests to the $\mathcal{R}$ can be calculated by using the following relation,

$$N_{ik}^n - N_{ik}^{n+1} \;\leq\; N_{ik}^{\mathcal{A}} + N_{ik}^n(1-\alpha_i), \tag{5.2}$$

where $N_{ik}^{\mathcal{A}}$ determines the number of segments read-ahead in the earlier intervals. Calculation of $N_{ik}^{\mathcal{A}}$ is detailed ahead. The lower bound of N_{ik}^{n+1}, denoted by L_{ik}^{n+1}, in 5.2 is $N_{ik}^n\alpha_i - N_{ik}^{\mathcal{A}}$. Note that the actual number of segments read N_{ik}^{n+1} could be greater than L_{ik}^{n+1}. The duration of each round i servicing request R_i is calculated as

$$\tau_{ik} = L_{ik}^{n+1} \cdot \lambda_i^{max}. \tag{5.3}$$

Now, if $\Sigma_i^{n+1}\tau_{ik} > T$, then the new request R_{n+1} is not admitted. If the total duration of all rounds is less than the duration of the interval, then the difference in durations is evenly distributed over all the rounds in the interval. This additional interval to each round defines the amount of read-ahead for each request R_i and is obtained by

$$\triangle t = \frac{T - \Sigma_{i=0}^{n+1}\tau_{ik}}{n+1}. \tag{5.4}$$

The read-ahead for request R_i is now defined as $N_i^A = N_{ik}^{n+1} - L_{ik}^{n+1}$. Initially, $N_{ik}^A = 0$ for all i.

Let s_i denote the size of each request $R_i \in \mathcal{R}$ remaining to be read from the disk. The maximum number of rounds required for each request $R_i \in \mathcal{R}$ is $\frac{s_i\lambda_i^{max}}{d}$, where $d = \min\{\tau_{ik} \mid 1 \leq i \leq n\}$. The value of m would then be

$$m = \max\{\frac{s_i \lambda_i^{max}}{d} \mid 1 \leq i \leq n\} \tag{5.5}$$

A simple admission control policy would adopt a "greedy" stance, calculating the buffer requirement for each new request and then admitting the request if the total buffer consumption in next m successive intervals would not exceed the maximum buffer size B_{max} and the QoS specifications (set by equation 5.2) of all the current clients requests would not be violated. Under such conditions, the goal of the admission control algorithm would be to maximize the number of client requests admitted. The greedy admission control algorithm takes in as input a request $(\{R_1, \ldots, R_n\}, R_{n+1})$ which consists of a set of requests being currently serviced, the new request R_{n+1}, and the QoS parameter for each client request and produces as output an admission decision for the new request.

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

$m \leftarrow \max\{s_i \lambda_i^{max}/d, \quad 1 \leq i \leq n\}$
for $i = 1$ **to** m **do**
  //Check if QoS specifications of each client is violated
  for $k = 1$ **to** n **do**
    if $N_{ik} - N'_{ik} > \ N^{\mathcal{A}}_{ik} + N_{ik}(1 - \alpha_i)$
    then return FALSE
  //Calculate the maximum buffer in each interval
  $B^i_{shar} = \max\{\sum_{j=1}^{n+1} B_{ji}(t^{jl}_{max}, t^{ji}_{c_s}) \mid 1 \leq l \leq n+1\}$
  if $B^i_{shar} > B_{max}$ **then return** FALSE
endfor
return TRUE

Buffer sharing increases the number of client requests admitted when compared with non-shared buffer schemes, since more clients can be admitted without overflowing the buffer. We offer some results in the next chapter that show the benefit of buffer sharing at both client and server sites.

Chapter 6

Implementation and Experimental Results

This chapter will detail our implementation and present our experimental results based on the approaches proposed in the previous sections.

6.1 System Implementation

6.1.1 Multithreaded Systems

A thread is a user level entity and is an independent sequence of execution of program code inside a Unix process [KSS96, LB96]. A traditional Unix process has a single thread of control that has sole possession of the process' memory and other resources. The threads share all of the process' memory. When one thread writes to the memory, other threads can read the results. The threads also share all the process' resources like file descriptors. Each thread has its own stack, but the code it executes is not part of the thread.

The client-server architecture consists of multimedia servers and multimedia clients. The server may get multiple requests from clients simultaneously. These requests must be handled serially if the server has only one thread of control. This can lead to service delays with long requests consuming a lot of service time. Multithreaded servers can be more adaptive to variations in user demands than the single-threaded servers, because each client request can be handled independently based on its priority. Long requests or requests of low priority do not block the servicing of other requests.

Another advantage of using multithreaded systems is conservation of memory space. Applications using more than one process and communication between its processes would require setup and extensive use of system resources. Whereas in a multithreaded application resource sharing is done implicitly through memory that is shared by all the threads. Also, programs using more than one thread of control need to maintain full address space and operating systems state for each process. The process of creating and maintaining this large amount of state makes each process much more expensive than a thread in both time and space. In addition, the inherent separation between processes may require a major programming effort to communicate between the threads in different processes or to synchronize their actions [KSS96]. Some of the other advantages of multithreaded systems are [LB96]

- Performance gains from multiprocessing hardware. Multiple threads are an efficient way for application developers to exploit the parallelism of the hardware.
- Increased application throughput. Multiple threads can run in an asynchronous manner maximizing processor utilization.
- Increased responsiveness. Computational intensive and otherwise long operations can be run on different threads thereby allowing the application to remain active and making the application more responsive to the user.

6.1.2 Multimedia Server

The multimedia server, henceforth referred as the server, sits on top of an object-oriented database system. The server is an event driven process that uses a time-ordered prioritized query queue to service requests.

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 server has three components, *viz* 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.

Each component in the server is multithreaded. The design of the server is illustrated in Figure 6.1. The communication module in the server has a `communication` thread, a `socket` thread, and multiple `service` threads. The communication thread creates a socket with `SO_REUSEADDR` for communication with the clients. This allows the address supplied by the `bind` call to be reused thereby making asynchronous access to the server by multiple clients possible. The communication manager then starts the socket thread that accepts connections on the socket created by the communication manager. Upon the arrival of a new connection request, the socket thread extracts the first connection on the queue of pending connections, creates a new socket with the properties of the existing socket, and allocates a new file descriptor for the new socket. This file descriptor is then sent to the communication manager via a pipe. The communication manager then starts a service thread to service the new request using the file descriptor sent to it by the socket thread and adds the new file descriptor to a list of active file descriptors. It then polls on the list of open file descriptors for input from the clients.

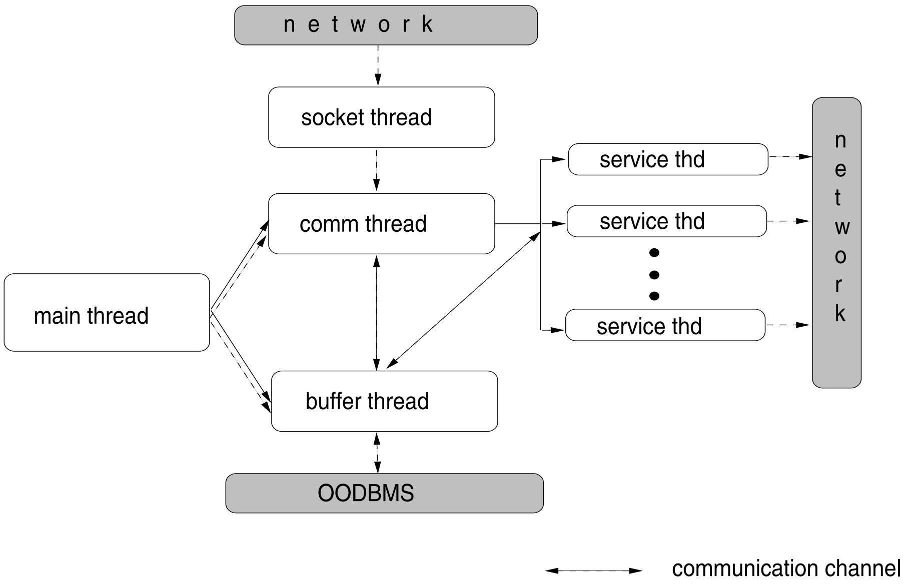

Figure 6.1: Design of the multimedia server

On receiving a query from a client at one of the open file descriptors, the communication thread stops polling and writes the query to pipe between the communication manager and the buffer manager. The buffer manager is simply another thread of control in the server. The retrieval of media data and storage of data in the buffer is the responsibility of the buffer manager. The `buffer` thread then proceeds to strip the query and fetch the appropriate segments of data from the disk into the local buffer. After loading sufficient amount of data into the buffer, the buffer thread informs the appropriate service thread to go ahead and read data from the buffer. Mutual exclusion on the shared buffer is maintained by using a set of semaphores between the service threads and the buffer thread.

A `service` thread waits on the buffer semaphore. As soon as data becomes available in the buffer, it reads the data and packs it to packets of equal size and writes them to the socket created by the socket thread. The size of data is sent in the first packet. The size of each packet is determined by the rate at which a client requests data. At present the server and the clients sit on a LAN. The client controls the rate at which the server must transmit data to the client. The client sets the rate at the beginning of each connection and can change it at the beginning of a new request.

6.1.3 Multimedia Client

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

The 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 presented. The scheduler is responsible for synchronizing the delivery of objects in all the participating media streams. Synchronization between different media streams is implemented using barriers. The buffer manager, as before, deals with the management of data in the buffer. Figure 6.2 shows the logical layout of the client.

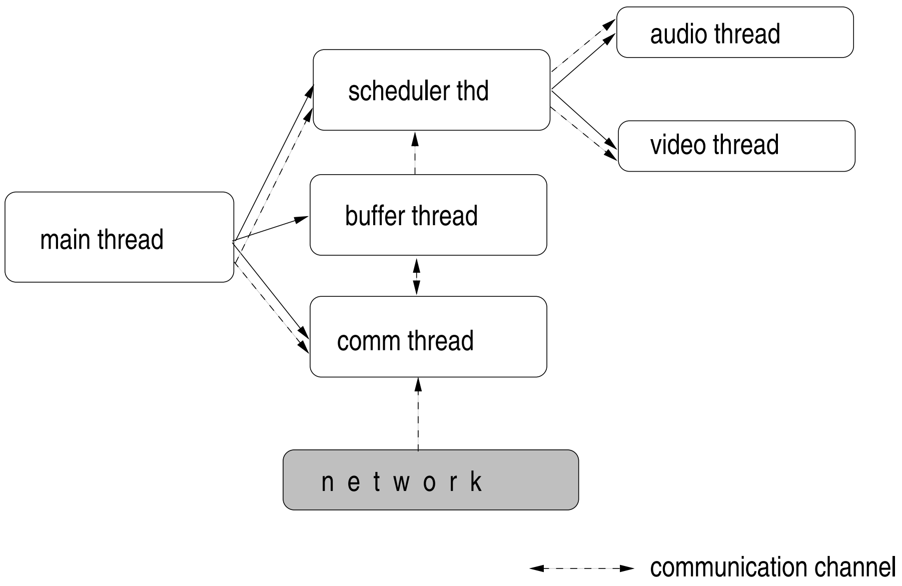

Figure 6.2: Design of the multimedia client

A presentation script as shown in Figure 6.3 is presented at each client. Each tuple in the presentation script refers to $\langle start\_time, end\_time, host\_address \rangle$. An event list is created from this script, as described in Chapter 3. Based on the time stamp of each START event in this list, a query is generated. The `communication` thread receives the query at the appropriate time from the main thread. The communication thread then opens the connection to the appropriate server, writes the query to the open socket and blocks till it hears from the server. On receiving the data from the server, it signals the `buffer` thread to unmarshall the data into the appropriate media object. This signaling is done via a semaphore. After creating the object, the buffer thread inserts the object into the appropriate slot in the client buffer.

```
VidPres[] = {{"vid0",0,5,HOST1}, {"vid1",5,10,HOST1}, {"vid2",10,15,HOST1},
            {"vid3",15,20,HOST1}, {"vid4",20,25,HOST1}, {"vid5",25,30,HOST1},
            {"vid06,30,35,HOST1}, {"vid7",35,40,HOST1}, {"vid8",40,45,HOST1},
            {"vid9",45,50,HOST1}};
AudPres[] = {{"Aud0",0,5,HOST2}, {"Aud1",5,10,HOST2}, {"Aud2",10,15,HOST2},
            {"Aud3",15,20,HOST2}, {"Aud4",20,25,HOST2}, {"Aud5",25,30,HOST2},
            {"Aud6",30,35,HOST2}, {"Aud7",35,40,HOST2}, {"Aud8",40,45,HOST2},
            {"Aud9",45,50,HOST2}};
```

Figure 6.3: An example presentation script

The `scheduler` thread schedules the delivery of media objects at times described in the presentation schedule. The delivery of all audio objects is carried out by an `audio` thread while the delivery of video objects is done by a `video` thread. Note that there can be more than one video thread and no more than one audio thread at each client site.

6.2 Experimental Results

In our experiments we have video-clips, audio-clips and images available as database objects. The presentation script is stated in the application program that runs on top of the multimedia server. At script realization time, methods of the media objects are invoked at the respective points in time. The playout management is implemented in a multi-threaded environment in which the functions of the buffer manager, scheduler and the presentation of the media streams are managed by threads.

6.2.1 Playout Management

We measured five QoS parameters including average delay, speed ratio, skew, jitter, and utilization during the presentation of the three media streams; these streams consisted of audio, video, and images, respectively. Two sets of experiments are illustrated in this to show the effect of synchronization on the QoS of the presentation. One involves audio and image streams and the other involves video and audio streams. In the examples presented below, QoS parameters are measured at client sites.

In the first set of the experiments, as in a typical slide presentation, objects from the audio and image streams were presented together. Each presentation of an audio or image object is for a duration of five seconds followed by a time gap of two seconds. Figure 6.4(a) shows the defined presentation of both the media streams. The presentation without delay recovery is illustrated in Figure 6.4(b). Figure 6.4(c) shows the actual presentation using Algorithm 1. It was found that the image stream consistently lagged in the presentation, possibly due to the additional overhead involved in creating the image and writing it into a window. When delay recovery is not implemented, the delay is allowed to propagate through the presentation. However, the synchronization between the two streams is still maintained, since the synchronization constraints enforce the simultaneous presentation of objects from both the media streams.

For the interval (t_0, t_1) in Figure 6.4, the average delay for the nominal case is 0. In the scenario without delay recovery, the average delay is 1.0, 1.71, and 11 in the intervals (t_0, t_1), (t_2, t_3), and (t_3, t_4), respectively. These values are presented in the first row in Table 6.1. Similarly, the speed ratio in the interval (t_0, t_1) for the case without delay recovery is 0.89. This is because only two-thirds of the audio object 2 is presented in (t_0, t_1). In the interval (t_1, t_2), the speed ratio is 0.78. Since all the objects delivered are presented without object loss, the utilization ratio is 1 for all time intervals. Note that the delayed stream, that is, the image stream, does not require any frame duplication to pause its presentation. Hence the value of utilization is 1 and not greater than 1. Furthermore, no difference in presentation times between an audio and image objects is observed. Therefore, the skew is 0 for all the intervals.

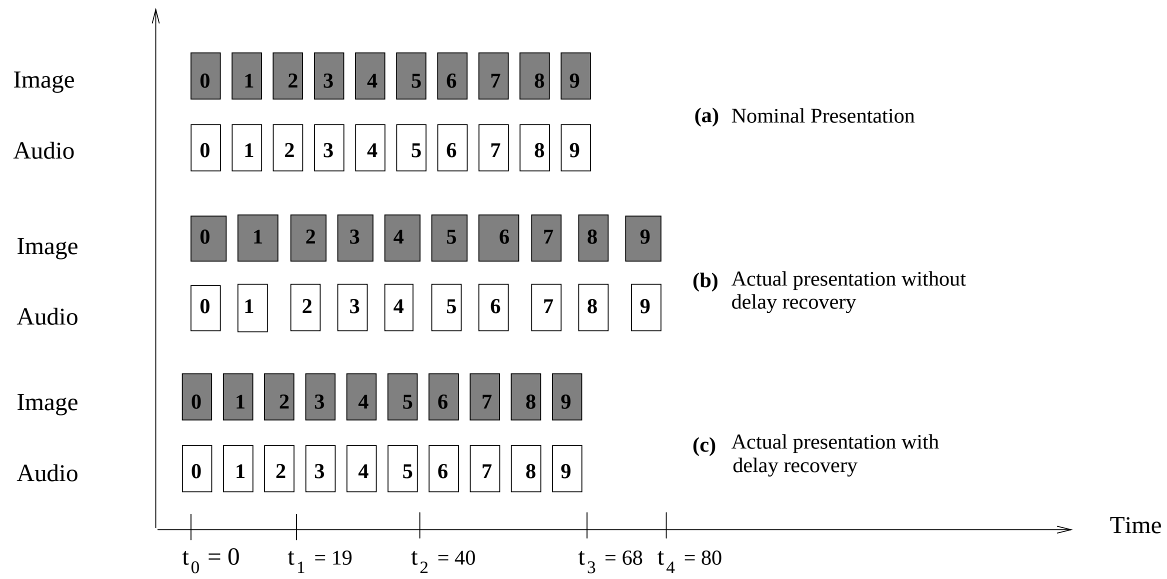

Figure 6.4: Presentation of audio and image streams : (a) nominal, (b) without delay recovery and (c) with delay recovery

The corresponding values when delay recovery is considered are shown in Table 6.2. The allowable skip for the image stream is set to a high value of 2 seconds, allowing the object to skip the entire delay. This is made possible by the non-temporal nature of images, so that information is not lost by decreasing the display time of an image object. Such a scheme results in the presentation schedule shown in Figure 6.4(c). Note that the utilization level does not decrease as a result of omission of the display of a non-temporal object. In contrast, if the delivery of an audio object were to be skipped or dropped, there would be a drop in the utilization level, as the actual data presented is less than the data made available. This difference is highlighted in the second set of experiments involving two temporal streams.

From Figure 6.4(c), one can clearly see that the average delay has been reduced to 0 for all intervals. The speed ratio for all intervals is equal to unity, as all necessary audio objects are presented in all intervals. Since all the temporal (audio) objects delivered are presented without object loss, the utilization ratio is 1 for all time intervals. As in the scenario without delay recovery, skew is observed to be 0 in all the intervals.

Parameter	$t_0 - t_1$	$t_1 - t_2$	$t_2 - t_3$	$t_3 - t_4$
Average Delay	1	1	1.71	11
Speed Ratio	0.89	0.78	0.9	-
Utilization	1	1	1	1
Skew	0/3	0/3	0/3	0/3

Table 6.1: Parameter values for presentation without delay recovery

Parameter	$t_0 - t_1$	$t_1 - t_2$	$t_2 - t_3$	$t_3 - t_4$
Average Delay	0	0	0	-
Speed Ratio	1	1	1	-
Utilization	1	1	1	-
Skew	0/3	0/3	0/3	-

Table 6.2: Parameter values for presentation with delay recovery

In the second set of experiments, we presented a video stream along with an audio stream. The video clips are MPEG encoded streams [SZ94, ZTSY95, ZLS95]. The nominal presentation schedule is shown in Figure 6.5(a). Unlike the slide presentation, the presentation of both the streams is continuous. The allowable skip for both the video and audio stream is set to 0.50 seconds. Similar to the previous case, QoS parameters were measured with and without delay recovery.

Tables 6.3 and 6.4 respectively present the QoS parameters for the case with and without delay

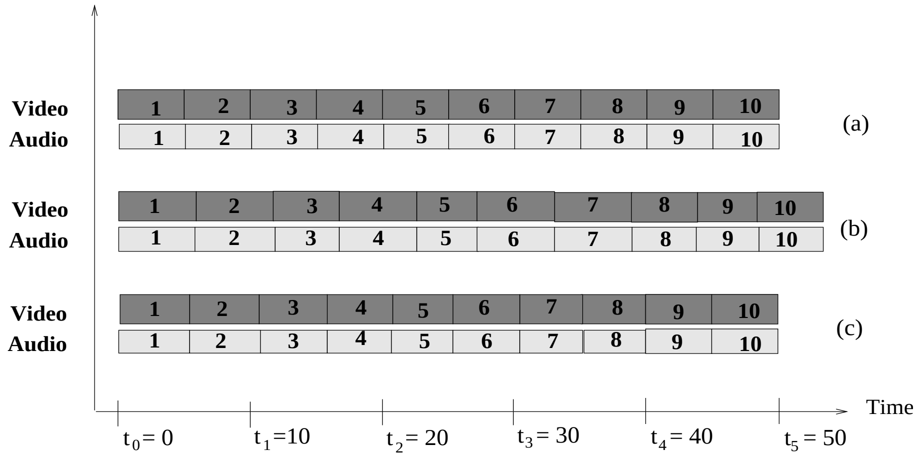

Figure 6.5: Presentation of audio and video streams : (a) nominal, (b) without delay recovery and (c) with delay recovery

recovery. Unlike in the previous case where the delays occurred in the image stream, presentation delays occurred in the both the streams. These delays were either due to system load and other extraneous conditions or the imposition of the presentation schedule on the delivery of a media stream. Since both audio and video are temporal in nature, we do see a decrease in the utilization as frames are dropped or skipped to maintain synchronization and decrease in average delay, skew and jitter in the presentation. The delay in the presentation could also be corrected by decreasing the speed ratio of the faster stream and thereby letting the slower stream catch up.

Parameter	$t_0 - t_1$	t_1	$t_1 - t_2$	t_2	$t_2 - t_3$	t_3	$t_3 - t_4$	t_4	$t_4 - t_5$	t_5
Average Delay	1.25	-	2.25	-	2.25	-	3.5	-	3.0	-
Speed Ratio	0.83	-	0.87	-	0.90	-	0.90	-	0.94	-
Utilization	1	-	1	-	1	-	1	-	1	-
Skew	0.0	-	0.5	-	0.5	-	1.0	-	0.0	-
Jitter	-	1.0	-	1.0	-	1.0	-	1.0	-	0.0

Table 6.3: Parameter values for presentation without delay recovery

For the interval (t_0, t_1) in Figure 6.5, the presentation of the video object was delayed while the presentation of that of the audio object was on time. In the absence of delay recovery, this delay was allowed to propagate to the following presentation intervals. This made the average delay substantial in each interval. For example, in intervals (t_0, t_1) and (t_1, t_2), the average delay was 1.25 and 2.25 seconds respectively. In contrast, with delay recovery, audio and video objects can be skipped as long as the skip is within the tolerable range for the presentation. This reduces

Parameter	$t_0 - t_1$	t_1	$t_1 - t_2$	t_2	$t_2 - t_3$	t_3	$t_3 - t_4$	t_4	$t_4 - t_5$	t_5
Average Delay	0.62	-	1.12	-	0.75	-	0.25	-	0.0	-
Speed Ratio	0.90	-	0.92	-	0.97	-	1.0	-	1.0	-
Utilization	0.95	-	0.95	-	0.95	-	0.90	-	1.0	-
Skew	0.0	-	0.25	-	0.25	-	0.0	-	0.0	-
Jitter	-	0.5	-	0.5	-	0.0	-	0.0	-	0.0

Table 6.4: Parameter values for presentation with delay recovery

the average delay to 0.62 and 1.12 seconds in (t_0, t_1) and (t_1, t_2) respectively. Table 6.4 illustrates this trend. Besides, speed ratio also increases when delay recovery is performed since the actual presentation rate approaches the nominal presentation rate. Skew was observed in all intervals except (t_0, t_1) when there was no delay recovery in place. This skew was reduced or eliminated when delay recovery was incorporated in the scheduling strategy. Without delay recovery, none of the video frames or audio samples were dropped. Hence, the utilization was considered 1 across the presentation with no delay recovery. However, with delay recovery some portions of the delayed stream, either audio or video or both, are skipped. This leads to a decrease in the utilization in all the intervals as shown in Table 6.4. In the case without delay recovery, jitter was observed at times $t_i, i = 1, \ldots, 4$ in the presentation. Using delay recovery jitter is reduced to 0.5 seconds at times t_1 and t_2 and is eliminated at t_3, t_4 and t_5.

From the experiments performed, we observed that although our scheduling algorithms minimize jitter and skew by invoking the synchronization events simultaneously, the system-level differences between two invoked participants may still result in the occurrence of jitter and skew. By using delay recovery, such occurrence can be reduced, as the amount of objects to be dealt with is reduced.

A playout management functionality allows the user to define an application specific presentation script. Moreover, the service greatly enhances the user's ability to generate acceptable schedules at various levels of granularity of synchronization and choose the right schedule.

We have shown that the implementation of the proposed synchronization algorithms minimizes jitter, skew and delays in a presentation. Furthermore, a buffer management strategy is also presented along with the synchronization algorithm. This allows us to minimize the average delays in the presentation, thereby increasing speed ratio.

6.2.2 Server Utilization

In another set of experiments, we measured QoS for each client by varying the number of clients to be admitted versus the server utilization. We fixed the buffer size at the server and increased the number of clients to be admitted. Note that the server utilization increases as the number of clients admitted increases. This, however, has a cost. The QoS of each client presentation starts to decrease as the number of clients increases. This result implicitly sets the bound on the number of clients that can be admitted before the QoS parameters of one or more of the current client requests are violated. Hence, admitting clients with an aim to increase server utilization does not always produce the desired result. Decreasing the QoS of the failed client request is a straight-forward way to admit the latest client request. Another way would be to reduce the QoS to all the existing client requests so long as the minimum QoS for any client is not violated. This approach adopts a fairness-to-all strategy. Figure 6.6 shows the effect of the number of clients on the server utilization. Currently, we are developing an admission control strategy that incorporates the QoS specifications of each client presentation, the network bandwidth and the server cache availability.

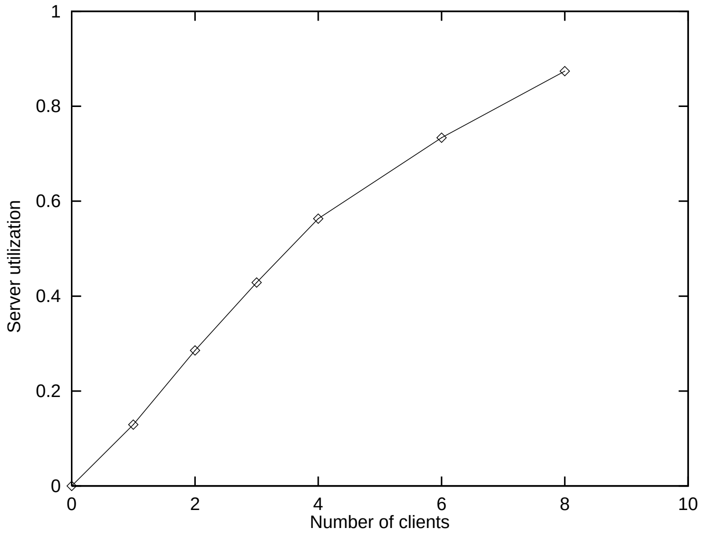

Figure 6.6: Server utilization vs. number of clients

We also measured the effect of buffer size at both the client and the server sites on server utilization. The consumption rate at the client site was around 8 KB/s for the audio stream and 46.5 KB/s for the video stream. The transfer rate across the network was around 500 KB/s. Figure 6.7 shows the increase in server utilzation with increase in the number of buffers at the client site. This increase does not continue indefinitely. The time for which the server is servicing a particular client is bounded by the time at which the last request is generated by the client. The effect of buffer size at the server on the server utilization is minimal.

6.2.3 Benefits of Buffer Sharing

In one set of simulations, we measured the effect of loading rate and the size of the object on the latency in a presentation. In the absence of any delays, we set the size of the first object to be

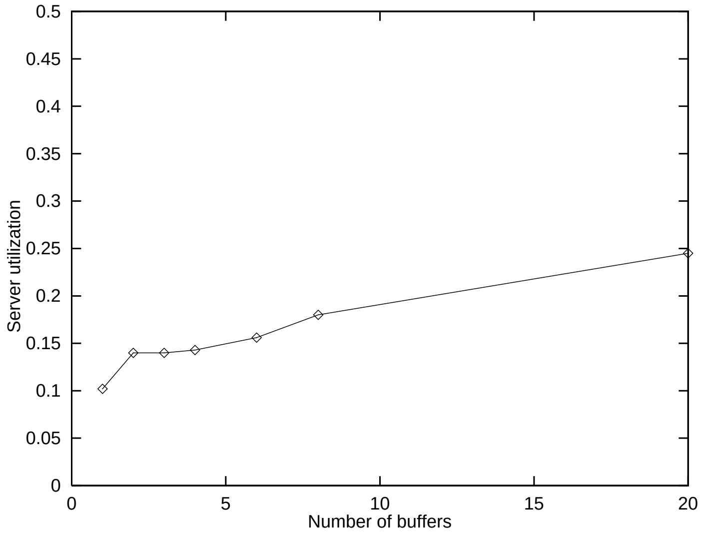

Figure 6.7: Effect of buffer size on server utilization

500 KB and the consumption rate to be 250 KB/s. The start of loading of the first object was set to zero. We then used (3.16) to determine the latency for various values of the loading rate. Figure 6.8 shows the effect of loading rate on the latency in the presentation. In another case, we fixed the loading rate at 100 KB/s and varied the size of the first object from 10 KB to 10000 KB. We found that the latency increased with an increase in size of the object. Figure 6.9 illustrates this trend.

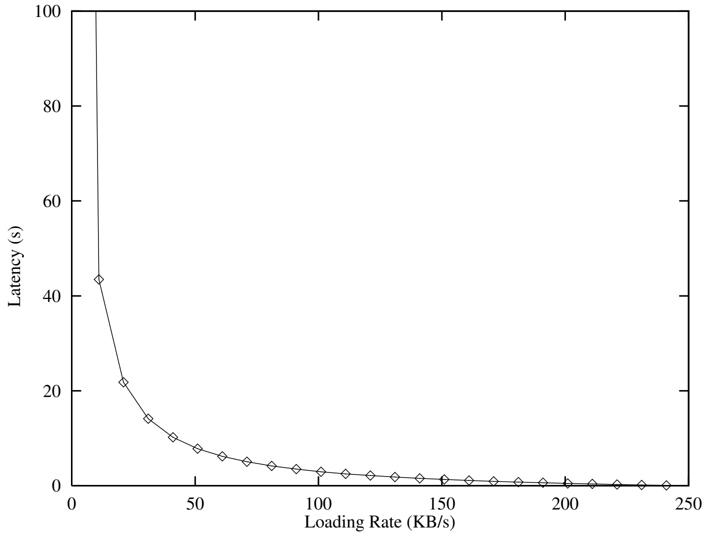

Figure 6.8: Effect of loading rate on latency

We simulated two situations to measure the effectiveness of buffer sharing at the client site. In the first situation, we fixed the consumption rate of all queries to be the same and set it to 250 KB/s (This is the consumption rate of a JPEG encoded stream displayed at 24 frames per second).

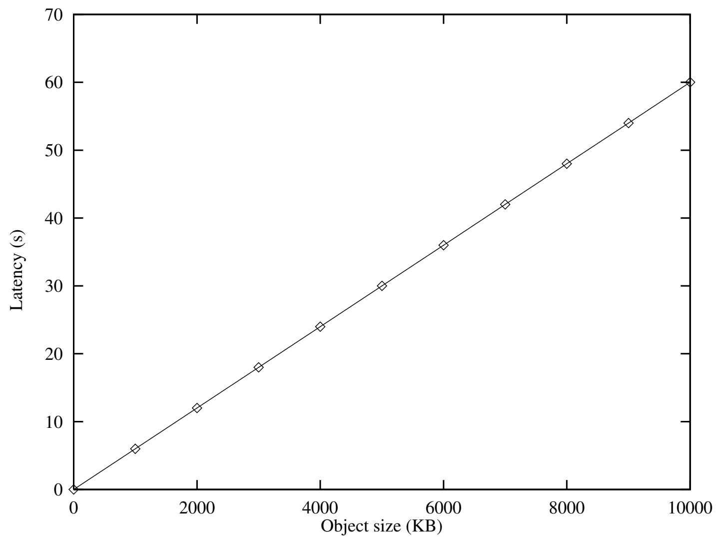

Figure 6.9: Effect of object size on latency

The loading rate was fixed at 500 KB/s (average transmission rate on an Ethernet). The length of the presentation interval was 2 seconds. The size of each media segment was 500 KB. Figure 6.11 shows the minimum buffer space required as the number of concurrent streams increased from 2 to 10. The loading of the ith media stream begins at time $t + \frac{2(i-1)}{n}$, where n is the number of streams presented in the interval $[t, t+2]$. Figure 6.10 illustrates the buffer consumption profile involving three media streams.

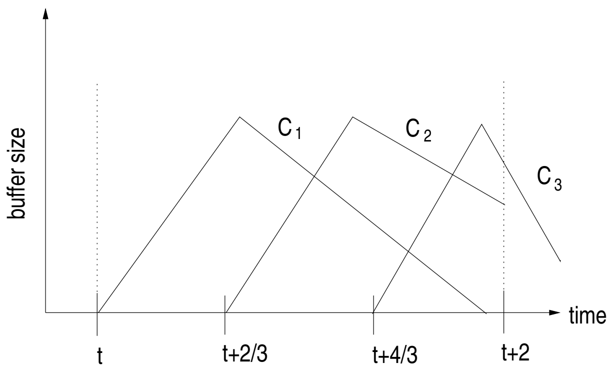

Figure 6.10: An example simulation

We find that the savings in buffer size with buffer sharing is close to 50%. Ng and Yang [NY96] reported this to be the maximum buffer reduction possible using buffer sharing. In the second case, we randomly set the consumption rates of streams in the range [100, 500] KB/s and calculated the minimum buffer requirement in the interval. The rest of parameters were set to the same values as in the first case. We found that buffer sharing reduced the buffer requirement by 30% to 50%. Figure 6.12 shows the buffer size with and without buffer sharing for this case.

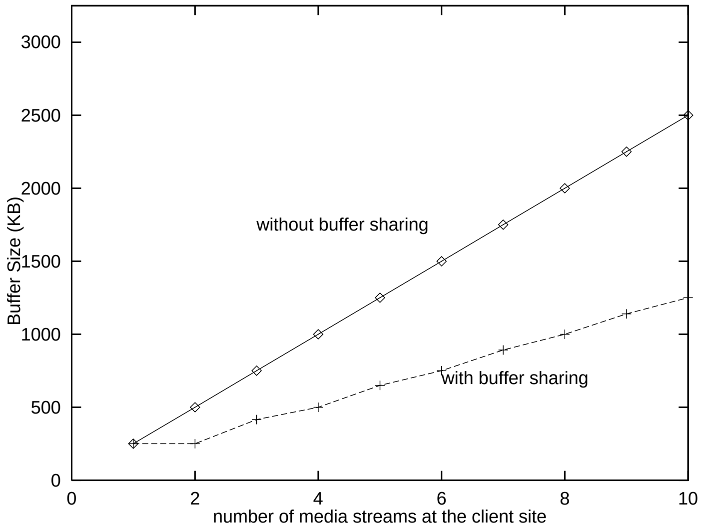

Figure 6.11: Benefit of buffer sharing with same consumption rates at the client site

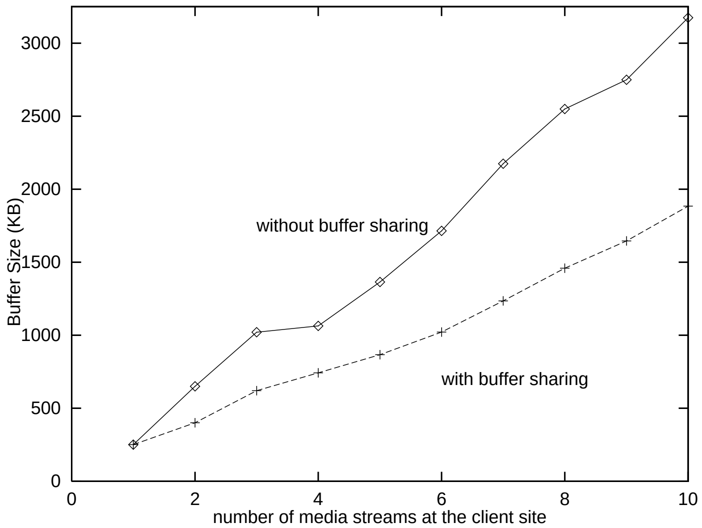

Figure 6.12: Benefit of buffer sharing with different consumption rates at the client site

We then carried out the same simulations at the server site to measure the benefit of buffer sharing. In the first case, the consumption rate was fixed at 500 KB/s while the loading rate was fixed at 2000 KB/s. The size of each interval was fixed at 500 ms. The graph in Figure 6.13 shows the minimum buffer space needed when the number of client requests varied from 2 to 10. We found that the minimum buffer required using buffer sharing was consistently 20% to 30%. In the second case, the transfer rate across the network was increased from 200 KB/s to 750 KB/s, thereby increasing the consumption rate. The minimum buffer requirement decreased with buffer sharing by 6% at a consumption rate of 200 KB/s and by 20% at a consumption rate of 750 KB/s. Figure 6.14 illustrates this trend.

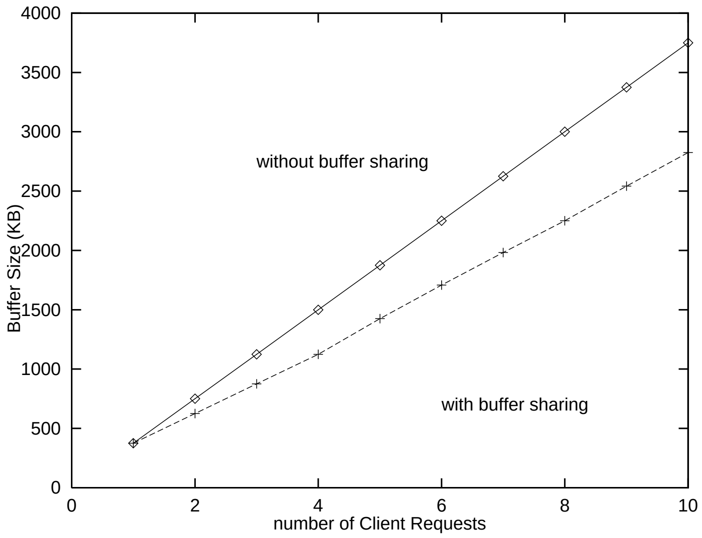

Figure 6.13: Benefit of buffer sharing with different consumption rates at the server site

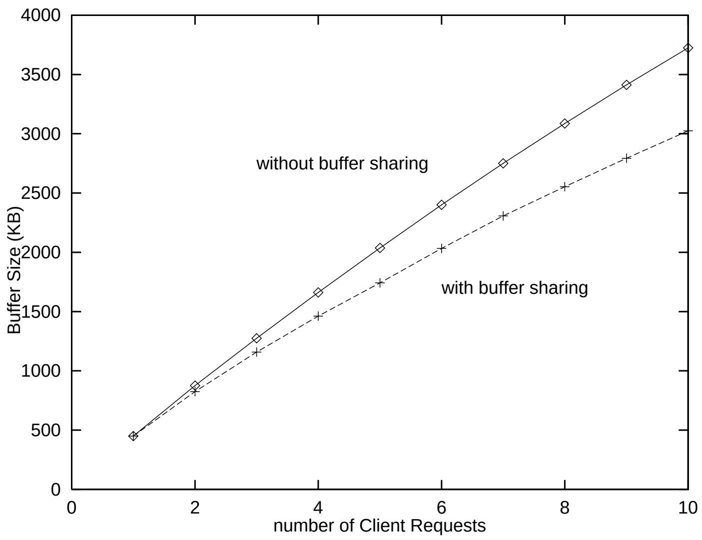

Figure 6.14: Benefit of buffer sharing with different consumption rates at the server site

Chapter 7

Conclusions and Future Work

Motivated by a need to allow users to access multimedia data freely across the network, we have developed a prototype distributed multimedia system and presented various issues like synchronization and buffer management at the client and server sites in the system. The work and possbile extensions are summarized in this chapter.

7.1 Multimedia Transaction Management

We introduced a framework for multimedia transaction management in database systems. This framework includes a vehicle for the specification of multimedia data, transactions, and schedule, and scheduling principles and algorithms to ensure synchronous presentations multiple media streams. A multimedia transaction consists of a set of transactions upon which synchronization dependencies are specified on the delivery operations to enforce both intra- and inter-synchronization constraints. A schedule of the multimedia transaction is acceptable only if it satisfies the synchronization constraints defined on the multimedia transaction with the allowable deviations. Several parameters, inluding average delay, speed ratio, utilization, jitter, and skew are used to set up permittable deviations. The proposed approaches have been implemented in a prototype that integrates the functionality of multimedia transaction management into an object-oriented database system.

In event of user interactions, synchronous presentations can also be restored by skipping/pausing individual media streams. Given both maximum permitted delivery delay and skipping intervals for the the participating media streams, a scheduling criterion should allow limited deviations in the synchronization dependencies and time constraints defined on a multimedia transaction. A precise formulation of such a criterion is to be investigated.

7.2 Buffer Management

A framework for buffer management in multimedia database systems was developed. This framework includes the principles of minimum consuming start times of media objects following their loading times, while guaranteeing the continuity and synchrony of the presentation of media objects, and the optimal buffer space requirements for media objects based on the least upper bounds on media objects consumption. A prefetching technique which satisfies the proposed buffer management principles is presented.

We have not addressed issues relevant to buffer management in event of user interactions. Given both maximum permitted delivery delay and skipping intervals for the the participating media streams, a buffer management strategy is needed to provide the most efficient buffer consumption and allocation while supporting user interaction such as fast forward and backward, skip, and pause.

7.3 Multimedia Server Functionalities

We have investigated the design and implementation of a distributed multimedia database system. This system is adaptable in that the quality of service provided can respond to changes in user quality requirements. Such adaptability is supported on the server side through tools which permit the specification of adjustments in the requested quality of service. We have introduced a best-effort approach to incorporate adaptability into server functionalities, including client scheduling, data buffering, and admission control. Through this approach, the system dynamically adjusts its state to best service client requests as specified in the QoS requirements. 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.

A more robust admission control strategy needs to be developed. Such a strategy would take into consideration the QoS requirements of each client, the network bandwidth, and the availablility of shared resources such as the server cache.

Bibliography

[AD90] A.Dan and D.Towsley. An Approximate Analysis of the LRU and FIFO Buffer Replacement Strategies. In *ACM SIGMETRICS*, pages 143–152, Denver, CO, May 1990.

[AGM88] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation. In *14th international conference on very large data bases*, pages 1–12, August 1988.

[AH91] David P. Anderson and George Homsy. A Communication Media I/O Server and Its Synchronization Mechanisms. *IEEE Computer*, 24(10):51–57, 1991.

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

[BMHD89] A.P. Buchmann, D.R. McCarthy, M. Hsu, and U. Dayal. Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control. In *Proceedings of the 5th Internatonal Conference on Data Engineering*, pages 470–480, 1989.

[CGS95] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi. Avoiding Retrival Contention for Composite Multimedia Objects. In *Proceedings of the 21th VLDB conference*, Zurich, Swizerland, 1995.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, and S. Sarin. The HIPAC Project: Combining Active Databases and Timing Constraints. *SIGMOD Record*, 17(1):51–70, March 1988.

[DDI+95] Y.F. Day, S. Dagtas, M. Iino, A. Khokhar, and A. Ghafoor. Spatio-Temporal Modeling of Video Data for On-Line Object-Oriented Query Processing. Technical report, School of Electrical Engineering, Purdue University, 1995.

[GC92a] 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.

[GC92b] J. Gimmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 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.

[GR93] Shahram Ghandeharizadeh and Luis Ramos. Continuous Retrievals of Multimedia Data Using Parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, 1993.

[GZ96a] 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.

[GZ96b] Sreenivas Gollapudi and Aidong Zhang. NetMedia: A Client-Server Distributed Multimedia Database Environment. In *Proc. of IW-MMDBMS '96*, Blue Mountain Lake, New York, August 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.

[KSS96] Steve Kleiman, Devang Shah, and Bart Smaalders. *Programming with Threads*. Prentice Hall, New Jersey, 1996.

[LB96] Bil Lewis and Daniel J. Berg. *Threads Primer*. Prentice Hall, New Jersey, 1996.

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

[LG90a] Thomas D.C. Little and Arif Ghafoor. Network Considerations for Distributed Multimedia Object Composition and Communication. *IEEE Network Magazine*, pages 32–49, 1990.

[LG90b] Thomas D.C. Little and Arif Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Arears in Communications*, 8(3):413–427, April 1990.

[MKK95] Frank Moser, Achim Kraib, and Wolfgang Klas. L/MRP: A Buffer Management Strategy for Interactive Continuous Data Flow in a Multimedia DBMS. In *Proceedings of the 21th VLDB conference*, Zurich, Swizerland, 1995.

[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.

[NY96] R. T. Ng and J. Yang. An analysis of buffer sharing and prefectching techniques for multimedia systems. *ACM Multimedia Systems*, 4(6):55–69, 1996.

[RRK93] P. Venkat Rangan, Srinivas Ramanathan, and Thomas Kaeppner. Performance of Inter-Media Synchronization in Distributed and Heterogeneous Multimedia Systems. *Computer Networks and ISDN Systems*, 1993.

[RV91] P. V. Rangan and H. M. Vin. Designing File Systems for Digital Video and Audio. In *13th Symposium on Operating System Principles (SOSP'91). Operating Systems Review, Vol. 25, No. 5*, pages 81–94, 1991.

[RV93] P. Venkat Rangan and Harrick M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, 1993.

[SRL88] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Concurrency Control for Distributed Real-Time Databases. *SIGMOD Record*, 17(1):82–98, March 1988.

[Ste90] Ralf Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Arears in Communications*, 8(3):401–412, April 1990.

[SZ94] S.W. Smoliar and H.J. Zhang. Content-Based Video Indexing and Retrieval. *IEEE Multimedia*, 1(2):62–72, 1994.

[TCG+93] A.U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory, design, and Implementation.* The Benjamin/Cummings Publishing, 1993.

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

[VR93] H. M. Vin and P. V. Rangan. Desigining a Multi-User HDTV Storage Server. *IEEE Journal on Selected Areas on Communication*, 11(1):153–164, January 1993.

[ZF93] T.F. Znati and B. Field. A Network Level Channel Abstraction for Multimedia Communication in Real-time Networks. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):590–599, 1993.

[ZG95a] Aidong Zhang and Sreenivas Gollapudi. Multimedia Transaction Management in Database Systems. Technical Report CS-TR-95-44, Dept. of Computer Science, 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.

[ZLS95] H.J. Zhang, C.Y. Low, and S.W. Smoliar. Video Parsing and Browsing Using Compressed Data. *Multimedia Tools and Applications*, 1(1):89–111, March 1995.

[ZRS87a] W. Zhao, K. Ramamritham, and J.A. Stankovic. Preemptive Scheduling Under Time and Resource Constraints. *IEEE Transactions on Computers*, 36(8):949–960, August 1987.

[ZRS87b] W. Zhao, K. Ramamritham, and J.A. Stankovic. Scheduling Tasks with Resource Requirements in Hard Real-Time Systems. *IEEE Transactions on Software Engineering*, 13(5):564–576, May 1987.

[ZTSY95] H.J. Zhang, S.Y. Tan, S.W. Smoliar, and G. Yihong. Automatic Parsing and Indexing of News Video. *Multimedia Systems*, 2(6):256–266, January 1995.