Multimedia Transaction Management in Database Systems

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

Abstract

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. In this paper, we investigate a framework and systematic strategies for supporting the continuous and synchronized retrieval and presentation of multimedia data streams in a multimedia database system. Specifically, we will develop: (1) a practical framework for specifying multimedia transactions and schedules and the appropriate synchronization constraints between different media streams; (2) multimedia transaction scheduling principles underlying the synchronized presentation of multimedia data streams when delay effects are considered; and (3) pragmatic techniques for multimedia transaction scheduling to support the synchronized presentation of multimedia data streams in object-oriented database systems.

1 Introduction

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 must be specified and maintained. Such synchronization is usually termed *intra-synchronization*. However, if a composite data stream is composed of media objects from different media streams, additional complications may arise with the timing relationships that may exist among the different types of media data streams. Such media data streams may not be merged prior to storage in a database as such a merger will vastly compound the difficulties of retrieving component media. Thus, the synchronization of multiple media data streams, termed *inter-synchronization*, becomes an essential prerequisite to any successful multimedia database application. For these reasons, synchronization has been recognized as one of the central problems in multimedia system development [LG90b, Ste90].

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 retrival from a server to multiple destinations over integrated networks. Both research efforts are directed toward the synchroniztion 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.

This paper will propose a framework for supporting continuous and synchronized presentation of multiple media data streams in a multimedia database system. We primarily seek to develop

- A practical framework for specifying multimedia data and inherent synchronization requirements; and
- Scheduling principles for synchronized presentation of multiple media data streams.

In the proposed framework, we first formulate a multimedia data model at the transaction management level for the purpose of scheduling media data operations. This data model facilitates the specification of time constraints on media objects. We then present a multimedia transaction model which introduces the concepts of multimedia transactions and schedules. A multimedia transaction defines a synchronous presentation of media streams and a scheule 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 transactioni 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. Finally, We present our implementation and experimental results based on the proposed approaches. A prototype has been designed which integrates multimedia transaction management strategies into an object-oriented database management system.

The remainder of this paper is organized as follows. Section 2 introduces the multimedia transaction model. In Section 3, we discuss an effective approach to specify synchronization dependencies on multimedia transactions. In Sections 4 and 5, we present the scheduling principles and algorithms to ensure the synchronous presentation of media streams in the event of delays. Section 6 sets forth our initial implementation and experimental results. Concluding remarks and future directions are offered in Section 7.

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.

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

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

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.

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 subschedules, 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 subschedules must be synchronized. A scheduler must ensure the correct execution of a multimedia transaction. A schedule or subschedule 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.

3 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.

3.1 Ensuring Time Constraints

As indicated in the multimedia transaction model given in Section 2, 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

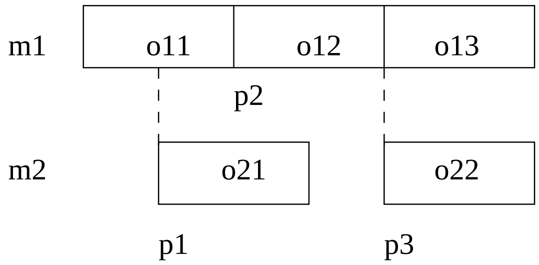

Figure 1: Synchronized streams.

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 $< o_{11}, 0, 1 >$, $< o_{12}, 1, 1 >$, $< o_{13}, 2, 1 >$ and m_2 consist of $< o_{21}, 0.5, 1 >$, $< o_{22}, 2, 1 >$. 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 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 experince demonstrates that such delays are frequent, the explicit specification and enforcement of synchronization constraints is necessary.

3.2 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 $< o_{ij}, t_{ij}, \triangle t_{ij} >$ $(1 \leq j \leq i_m)$. Without loss of generality, we assume that the synchronization constraints are implied in the definition of time 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. 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

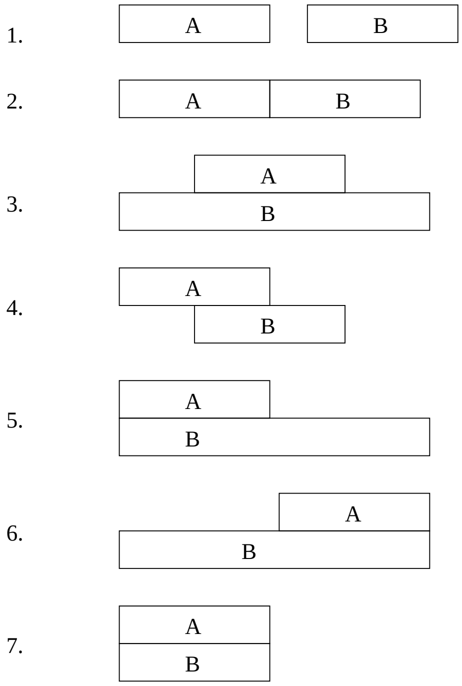

Figure 2: Typical temporal relationships.

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 $< o_{ij}, t_{ij}, \triangle t_{ij} >$ 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 $< o_{i1}, t_{i1}, \triangle t_{i1} >$ and $< o_{i2}, t_{i2}, \triangle t_{i2} >$ with o_{i1} starts before o_{i2} in a single media stream are *neighboring* each other if there is no object $< o_{i3}, t_{i3}, \triangle t_{i3} >$ 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_i has an intra-synchronization point p_1 if there exist two neighboring objects $< o_{i1}, t_{i1}, \triangle t_{i1} >$ and $< o_{i2}, t_{i2}, \triangle t_{i2} >$ such that $s_{o_{i2}} = e_{o_{i1}} + \triangle t$, where $\triangle t = t_{i2} - (t_{i1} + \triangle t_{i1})$.*

Definition 2 (Inter-synchronization point) *A media stream m_0 has an inter-synchronization point p_1 if there exists an object o_{0h_0} and a set of media streams with objects $< o_{1h_1}, t_{1h_1}, \triangle t_{1h_1} >, \ldots,$ $< o_{lh_l}, t_{lh_l}, \triangle t_{lh_l} >$ such that o_{0h_0} starts equal to o_{1h_1} and o_{1h_1} starts equal to o_{2h_2} and ... and o_{lh_l} starts equal to o_{0h_0} or o_{0h_0} ends equal to o_{1h_1} and o_{1h_1} ends equal to o_{2h_2} and ... and o_{lh_l} ends equal to o_{0h_0}.*

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

has an middle-synchronization point p_1 *if there exists another media stream* m_2 *with object* $< o_{2j}, t_{2j}, \triangle t_{2j} >$ *such that* $t_{1i} < t_{2j} < t_{1i} + \triangle t_{1i}$ *or* $t_{1i} < t_{2j} + \triangle t_{2j} < t_{1i} + \triangle t_{1i}$.

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 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*

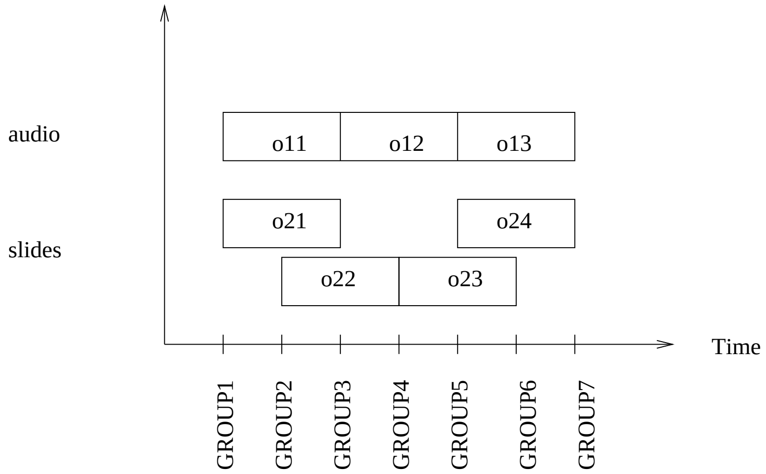

Figure 3: Layered GROUPs.

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 3. In this application, a set of layered GROUPs, denoted $GROUP_1, \ldots, GROUP_7$*, are identified.*

4 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 Section 3 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 acturally 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 permittable range.*

Note that the scheduler need not consider the concurrent execution of multiple multimedia transactions. There is no need to have a central schedular 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.

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 controling 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 permittable delay constraint that defines the maximum tolerable delay during the execution of multimedia transaction T. We assume that each media stream has a permittable delay constraint and the minimum value of all permittable 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, as follows:

Algorithm 1

Input: T, the multimedia transaction with l number of layered GROUPs.

Coordinator for a multimedia transaction

$i \leftarrow 1$

All START events in GROUP$_i$ are concurrently invoked

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 invoke the END events

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 all START events in the GROUP$_i$ are concurrently invoked.

return

Participant for an individual transaction

wait for START message from Coordinator

invoke delivery operation

wait for END message from Coordinator

send END message to coordinator

return

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.

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 Section 4, 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. The *average delay* is the average presentation delay of each object in a time interval. The *speed ratio* is the actual presentation rate over the nominal presentation rate. The *utilization* equals the actual presentation rate over the delivery rate. Ideally, both the speed and utilization ratios 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 Section 4, 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 Section 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 Section 4, 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:

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

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 Section 4:

(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.

Algorithm 2
Input: multimedia transaction T with l number of layered GROUPs; allowable SKIP and DELAY for media objects in T.

Coordinator with delay recovery

$i \leftarrow 1$

All START events in GROUP$_i$ are concurrently invoked

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 invoke the END events

wait for $e \leftarrow$ the first END event to complete

on timeout begin abort T; **return end**;

for the rest of END events **do**

WAIT $\leftarrow$ 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 $e \leq min(\{d_1^{max}, d_2^{max}, ..., d_l^{max}\})$

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 all START events in the GROUP$_i$ are concurrently invoked.

return

Participant for an individual transaction

wait for START message from Coordinator

invoke delivery operation

wait for END message from Coordinator

if receive STOP signal from Coordinator

then terminate delivery operation

send END message to coordinator

return

6 Implementation and Experiments

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

6.1 System Model

A multimedia playout management functionality was developed on top of O_2, an object-oriented database system. This functionality is integrated with the other services provided by the database system like transaction management, storage management, and concurrency control. One of the advantages of such an architecture is it provides 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 (see Figure 4). As everything is handled within the same system, efficient interplay between playout management components and other database management system components is possible.

As shown in Figure 4, the multimedia transaction manager contains two main modules: a multimedia transaction language (MTL) interpreter and a media manager (MM). The multimedia transaction language MTL interpreter allows users to specify a set of transactions associated with a multimedia transaction, including intra- and inter-synchronization requirements on component transactions. A multimedia transaction specified in MTL is then processed by the interpreter, and data accesses are sent to both the MM and the underlying O_2 DBMS for processing. Note that the design strategies can be applied to any OODBMS environment that support a C++ interface, such as ODE, an object-oriented database system developed at AT&T Bell Lab.

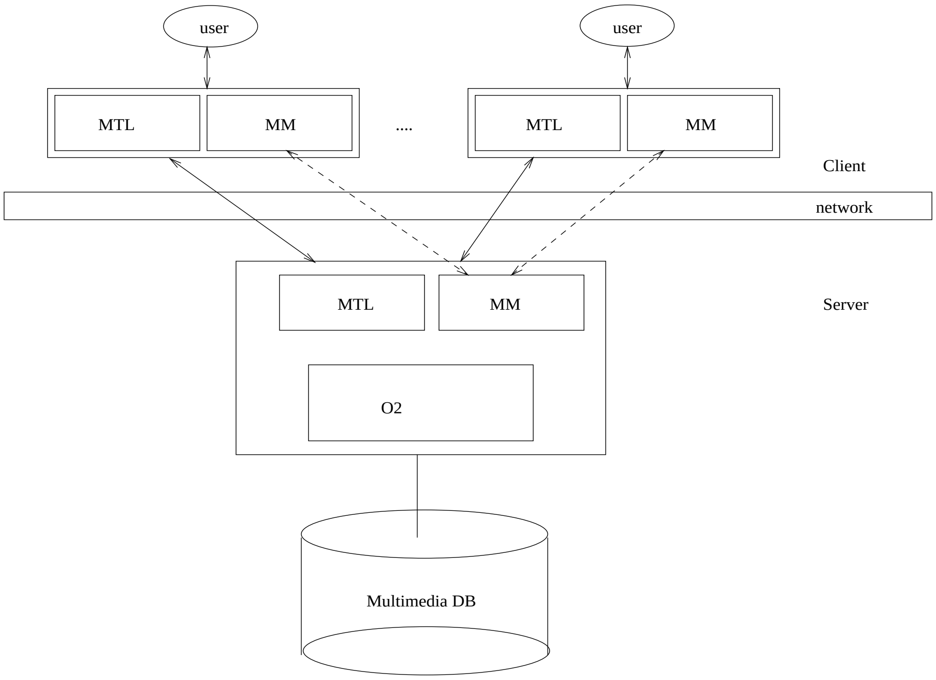

Figure 4: System model

Every object in the database is distinguished by the tuple $< o, t, \triangle t >$ where t is the starting time of object o. The finishing time can be obtained by adding the duration $\triangle t$ to the starting time. A media stream is represented by a list of tuples signifying the objects in the stream. A presentation involving media streams can be embodied by a group of these lists, each list representing a media stream. We consider an application involving the presentation of two media streams: audio and images.

6.2 The Process

We employed a model of distributed computation to enforce synchrony in the set of processes that deliver each object in the media stream. Each process delivers a media object and communicates with other processes by `sending` and `receiving` messages. In our model, a physical process p_{mi} has the following characteristics:

- p_{mi} delivers the object o_i from media stream m;
- p_{mi} has START and END event corresponding to the START and END events of o_i;
- p_{mi} is dependent on process p_{mj} if o_i is dependent on o_j;

- p_{mi} is dependent on all the processes whose END events are in the same GROUP as the START event of p_{mi};
- All processes whose END events are in the same GROUP as the START event of p_{mi} are dependent on p_{mi}.

Typically, a process p_{mi} waits until it receives messages from every process it is dependent on, delivers the object and then sends messages to all processes that are dependent on p_{mi} (see Figure 5). The messages are passed using the message passing facilities of UNIX.

receive message from scheduler to START
receive messages from all END events in GROUP
send message to scheduler signaling START
DELIVER MEDIA OBJECT
receive message from scheduler to END
send message to all START events in GROUP
send message to scheduler signaling END

Figure 5: Handling delivery of a media object

Every process communicates both with the scheduler and with other processes in the same group, ensuring that synchronization constraints are maintained in every GROUP. It suffices to maintain synchronous execution of START and END events in each GROUP to generate a synchronous schedule of presentation.

In order to reduce the number of processes present at any time in the system, a process is spawned only when an object needs to be delivered. Furthermore, the imposition of synchronization constraints on the presentation schedule translates into the generation of a process schedule. This, in turn, dictates when a process should start and end. The overlap of objects in the presentation of the image stream necessitates the spawning of as many processes as there are overlapped objects at any given time. Assuming the maximum number of object overlapping in a presentation of media streams to be l, the optimum number of processes required to present the media objects would then be l. Generating a synchronous schedule guarantees the use of an optimum number of processes to deliver the media objects. The presence of a less-than-optimum number of processes results in an incorrect presentation schedule and the aborting of the multimedia transaction. For example, consider a presentation schedule shown in Figure 6. Since there is an overlap of 2 in the image stream in all the intervals, the optimum number of processes required to present the image stream would be 2. If the number of processes is less than 2, presentation of image object 1 is impossible without violating the middle-synchronization point at t_1 which would lead to an incorrect presentation schedule. Note that even if the number of processes that present the image stream is greater than 2, there would be no change in the presentation schedule. The optimum number of processes in the entire presentation is 3.

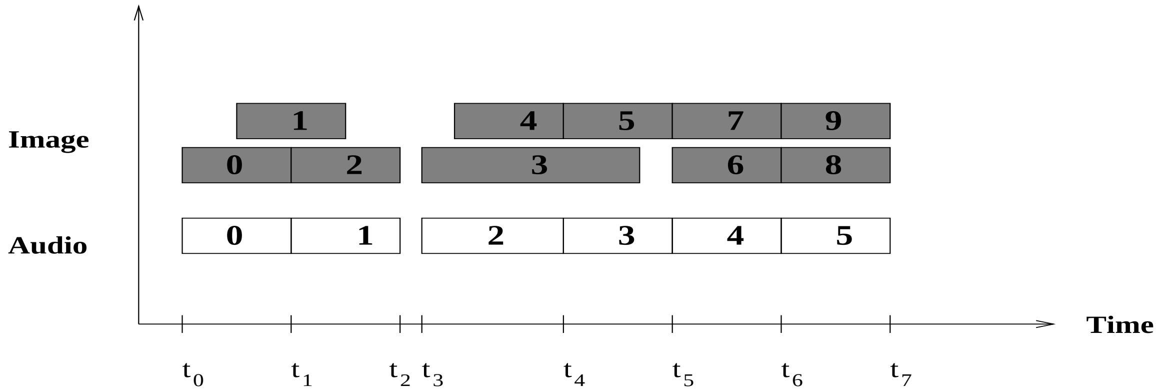

Figure 6: A presentation schedule

6.3 The Scheduler

A scheduler is used to control the invocation of the START and END events of a process. The scheduler invokes events in a groupwise fashion. That is, all the events in GROUP_i are invoked by the scheduler; these are followed by the events in GROUP_{i+1}. The invocation of a START event involves the following:

- Sending a message to the process p to which the START event belongs.
- After p gets a message from the scheduler to START, it blocks until all the END events in the same GROUP are completed. This ensures the maintenance of temporal ordering of events within a GROUP.
- After receiving messages from all END events in the GROUP, p sends a message to the scheduler signaling its START.

Similarly, the invocation of an END event include:

- Sending a message to the process p to which the END event belongs.
- After receiving a message from the scheduler, p sends messages to all the processes whose START events are in the same GROUP.
- p then sends a message to the scheduler signaling its END.

Thus the scheduler waits until it receives messages from all the processes whose START or END events are in GROUP_i before it sends messages to all the processes whose START or END events are in GROUP_{i+1}. Note that even though the scheduler has sent messages to the events in GROUP_{i+1}, they may not start immediately because (1) all END events in GROUP_{i+1} need to wait for the respective objects to be delivered and (2) all START events wait until the END events in GROUP_{i+1} finish first. Such a scheme saves time as the messages are sent to the process as the objects are being delivered. Any additional time incurred in delivering the messages is thus saved.

6.4 Delay Recovery

The presentation of multimedia streams are prone to delays, which may arise from causes ranging from system load to delivery schedules. Such delays can be circumvented by skipping the slower medium, pausing the faster medium, or a combination of the two. In the original scheduling algorithm, the scheduler simply waits for all the events in GROUP_i to be completed before invoking events in GROUP_{i+1}. This approach results in delay propagation through the entire presentation. That is, the slowest stream always delays the scheduling of the ensuing events in the presentation. In order to avoid this situation, the scheduler waits only until the first END event in GROUP_i completes. If this event is delayed, the scheduler calculates the delay. This delay is the inherent delay existing in the presentation. At this point, all the remaining (delayed) processes in GROUP_i send their respective delays to the scheduler and continue with the presentation of their media objects. The scheduler then calculates the difference between the delay $\triangle d$ and the allowable skip $\triangle s$ for each delayed object and waits for an additional period equal to the maximum difference. At the end of the wait, the scheduler sends STOP signals to all the delayed processes to stop the delivery of their objects. After receiving a STOP signal from the scheduler, a delayed process skips the rest of the presentation of its object and the sends a message to all the START events in GROUP_i. The process then sends back a message to the scheduler signaling its END. Thus, after all the delayed processes have ended their execution, the START events in GROUP_i would be invoked. Following the successful initiation of all the START events in GROUP_i, the scheduler then goes on to initiate events in GROUP_{i+1}. This way, the scheduler recovers delay between two successive synchronization points in the presentation.

6.5 Experimental Results

We measured four QOS parameters including average delay, speed ratio, skew, and utilization during the presentation of two media streams; these streams consist of audio and images, respectively. As in a typical slide presentation, objects from the two streams were presented together. Each presentation of an audio and image object is for a duration of five seconds followed by a time gap of two seconds. The whole presentation lasted 68 seconds. Figure 7(a) shows the defined presentation scheme of both the media streams. Figure 7(b) shows the actual presentation using Algorithm 1. We found the image stream consistently lagging in the presentation. This could be because of the additional overhead involved in loading the image and writing it into a window. In the case without delay recovery, the delay is allowed to propagate through the presentation. However, the synchronization between the two streams is still maintained. This is because the synchronization constraints enforce the simultaneous presentation of objects from both the media streams.

For the interval (t_0, t_1) in Figure 7, the average delay for the nominal case is 0. In the case without delay recovery, the average delay equals 3 divided by 3, or 1.0. For the interval (t_2, t_3), the average delay further increases to 6 divided by 3.5, or 1.71. In the interval (t_3, t_4), there is only 1 object being presented and its delay is 11. Therefore, the average delay in this interval is 11. These values are shown in first row in Table 1. Similarly, the speed ratio in the interval (t_0, t_1) for the case without delay recovery would be 2.67 (the number of objects actually presented) divided by 3 (the number of objects that should have been presented), or 0.89. Including partial objects in the calculation of presentation rate would be more meaningful for temporal media such as audio and video. In the interval (t_0, t_1), only two-thirds of the audio object 2 is presented. Therefore, the number of presentations are set to 0.89. Similarly, for the interval (t_1, t_2), the speed ratio would be 2.33 divided by 3, or 0.78. Since all the objects delivered are presented without the loss of any object, the utilization ratio is 1 in all the time intervals. Note that the delayed stream, being 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 object is observed. Therefore, the skew is 0 in all the intervals. We would like to mention that though the synchronization algorithm does minimize skew and jitter in the presentation, it does not eliminate them completely due to delays arising from system load.

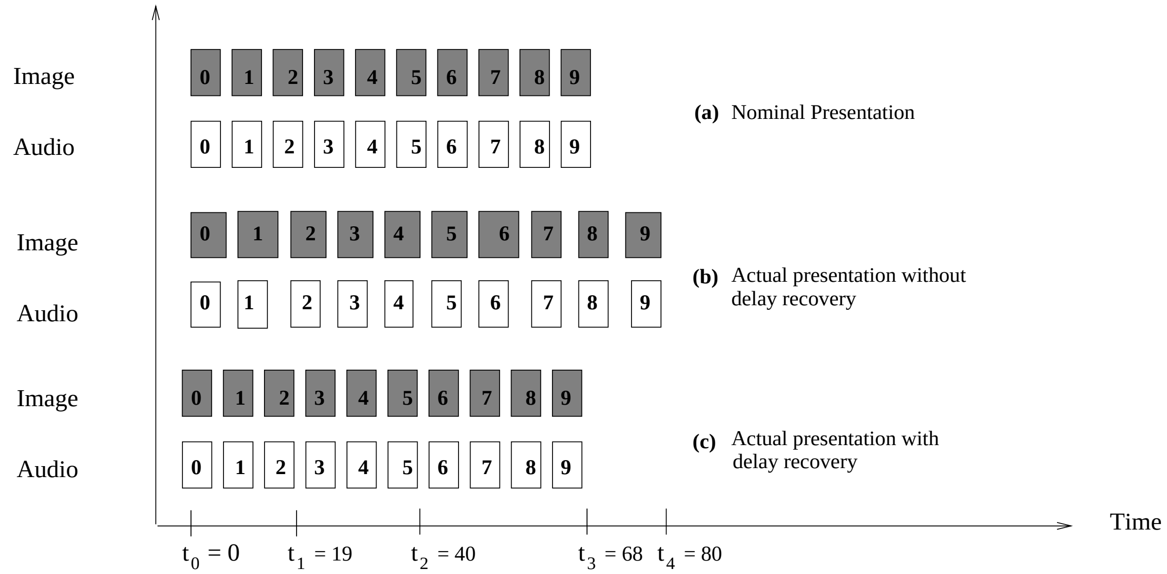

Figure 7: Presentation scheme of both the media streams : (a) nominal, (b) without delay recovery and (c) with delay recovery

The corresponding values when delay recovery is considered using Algorithm 2 are shown in Table 2. The allowable skip for the the image stream is set to a high value of 4, 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. Therefore, as soon as the non-delayed object (audio) ends, it sends a message to the process delivering the image object.

The process delivering the image object simply skips the rest of its delay and ends its display. Such a scheme results in the presentation schedule shown in Figure 7(c). Note that the utilization does not decrease because the display of a non-temporal object is being skipped. On the other hand, if the delivery of an audio object were to be skipped or dropped, there would be a drop in the utilization as the actual data presented is less than the data made available.

From Figure 7(c), one can clearly see that the average delay has been reduced to 0 for all intervals. The speed ratio in all the intervals is equal to unity as all audio objects that need to be delivered are presented in all the intervals. Since all the temporal (audio) objects delivered are presented without the loss of any object, the utilization ratio is 1 in all the time intervals. As in the case 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 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 2: Parameter Values for presentation with delay recovery

We have shown that the synchronization algorithm we implemented to maintain time and synchronization constraints minimizes skew and jitter in a presentation. Furthermore, an algorithm to handle delays in a presentation is implemented. This allows us to minimize the average delays in the presentation, thereby increasing speed ratio.

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.

7 Conclusions and Future Research

In this paper, we have 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, itter, 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. This implemetation is conducted using O_2, a well-known 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.

References

[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.

[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.

[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.

[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.

[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.

[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.

[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).

[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.

[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.