INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Design and Implementation of a Synchronization Scheduler In a Distributed Multimedia Presentation System

Shiyan XU

A Thesis

In

The Department

Of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montreal, Quebec, Canada

August 1999

Abstract

Design and Implementation of a Synchronization Scheduler

In a Distributed Multimedia Presentation System

Shiyan XU

A distributed multimedia document presentation involves retrieval of objects from one or more servers and their presentations at a client system. The retrieval and presentation of media objects have to be carried out in accordance with some temporal specification, i.e. temporal synchronization. In this thesis, we design and implement a synchronization scheduler in a distributed multimedia presentation system. First, we develop an interval constraint graph model to describe the temporal relationships among media objects. Second, based on the model, we give a way to produce an optimal schedule from a set of constraints using a linear programming solver. Third, we develop a run time rescheduling strategy to correct dynamic synchronization errors by means of a retiming method. Finally, we implement the scheduler in a simulated distributed environment to demonstrate its usefulness and effectiveness.

The interval constraint graph model we developed takes into account the QoS of temporal constraints of a multimedia document. Our flexible temporal model can help to derive an optimal schedule and carry out run time rescheduling. Then, a multimedia document can be presented with better qualities of temporal constraints, or leading to a better presentation.

Acknowledgements

I want to thank and acknowledge my supervisor, Dr. H. F. Li, for everything he did for me during the past two years. I believe that he has done all he could do to guide me to work on my thesis as a master student. I also want to thank Dr. R. Jayakumar. His help is so important to complete my thesis.

Table of Contents

List of Figures

CHAPTER 1 Introduction

Recent developments in multimedia technologies and broadband networks have significantly contributed to the emergence of new multimedia applications such as teleconferencing, services-on-demand, multimedia email, etc. A presentation in such multimedia applications involves spatial organization, temporal organization, and the retrieval of multimedia objects from distributed multimedia servers. This, in turn, allows users to interact with the presentation sequences as well.

A key aspect in a multimedia presentation system is temporal synchronization which defines the temporal dependencies among media objects. In terms of temporal synchronization, the presentation of multimedia objects can either be live or orchestrated. In a live presentation, we have simultaneous acquisition of voice and video, and hence the temporal relationships are implied and dynamically formulated. In an orchestrated presentation, the temporal relationships are explicitly formulated, as in the case of objects with voice annotated text. The media characteristics and temporal interdependencies must clearly be established for ensuring proper scheduling of the synchronized presentation.

In this thesis, we focus on the problems related to temporal synchronization in an orchestrated multimedia presentation system. The synchronization scheme should model the temporal inter-dependencies of media objects and ensure the scheduling of media synchronization.

1.1 Temporal Synchronization Models for Multimedia Presentations

In order to present multimedia objects, we need modeling techniques to organize and synchronize them. A major challenge for multimedia systems is how to synchronize various data types such as text, image, video, sound, and graphics, which possibly come from distributed sources to compose sophisticated multimedia presentations, especially at the presentation time. To satisfy the temporal requirements of multimedia data, some temporal constraints in a specification are defined to capture the relationships among the media objects to be presented. In addition, a proper specification model is needed for the description of temporal constraints. Use of such a model capable of representing temporal constraints on multimedia data makes it easy to satisfy these constraints at the presentation time.

To express temporal constraints among media objects, E.Bertino and E.Ferrari classify temporal models into two broad categories: timeline models and constraint-based models [BF98]. As of now, a large number of temporal models have been developed for multimedia presentations, but many of them do not consider synchronization requirements in distributed multimedia systems.

An interval-based model is a kind of constraint-based model in which the temporal constraints are expressed by the relationship between intervals [A83]. However, most of the interval-based models lack the quantitative aspect of time. Users of multimedia presentations are not only interested in the qualitative relationship between two events that refer to the occurrence of any action, but they are also interested in the quantitative relationship between them.

On the other hand, a very basic model that addresses the quantitative relationship needs of multimedia applications is the timeline model in which the time each event takes

place is expressed in absolute time. However, the timeline model is too rigid for many purposes, and the user must specify exactly when an event should occur. Hence the timeline model is only suitable for applications that do not require any flexibility in temporal specifications and do not need to be modified once they are created.

To overcome the inflexibility of temporal models, many methods are proposed, such as an interval-based model using object composition petri nets (OCPN) in [LG90], another interval-based conceptual model in [LG93], and a Time Flow Graph (TFG) based model in [KG94]. They are very useful in multimedia presentations because of their flexibility.

1.1.1 Firefly

A very useful constraint-based model for specifying temporal relationships among media objects within a document is provided by Firefly [BZ92] [BZ93]. Buchanan and Zellweger use constraints to specify temporal relationships in their system called Firefly. Their work is based on the use of difference constraints for the specification of temporal information.

Firefly provides a graph notation to express the temporal synchronization. It consists of square nodes, circular nodes and edges. Square nodes represent the starting and ending time of media objects. Circular nodes represent intra-object synchronization points and are placed between the start and end events of the corresponding object. Edges represent temporal relationships between events with a label. The examples of labels are *"simultaneous with"* and *"before by 10s"*.

An example of FireFly is shown in FIGURE 1, which models a multimedia presentation consisting of three objects obj_1, obj_2 and obj_3. In the example, the start of obj_1 ($t_{start}(obj_1)$) should occur at the same time with the start of obj_2 ($t_{start}(obj_2)$). The start of obj_3 ($t_{start}(obj_3)$) should occur after 2 seconds the end of obj_2 ($t_{end}(obj_2)$) happens. Similarly the rest of the temporal requirements are specified.

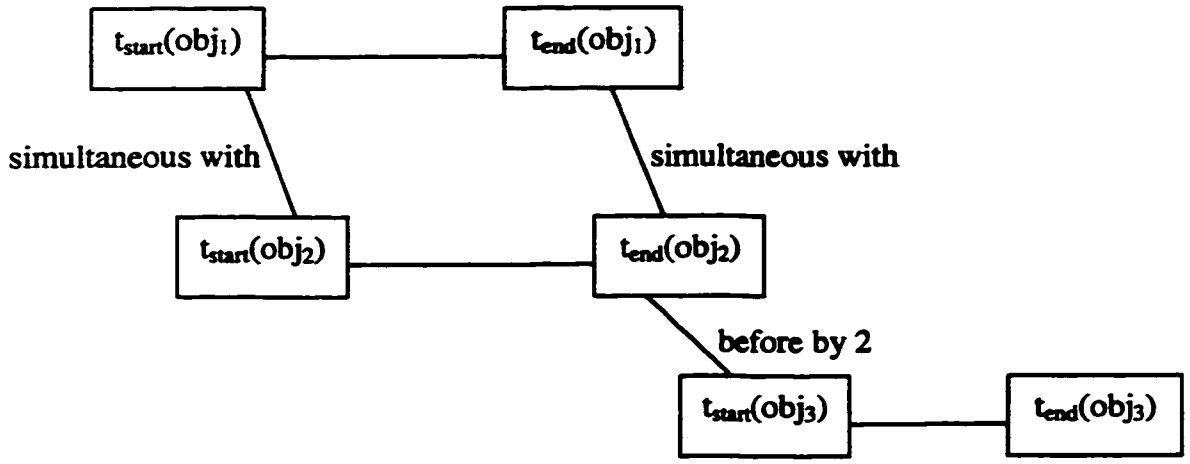


FIGURE 1: An Example of Firefly

This model supports two types of constraints: temporal equalities and temporal inequalities. The former requires either that two events occur simultaneously, or that one precedes the other by a fixed amount of time, while the latter supports the expression of indeterminacy.

As we can see, Firefly has a very strong ability to express both the qualitative temporal relationship and the quantitative relationship among media objects, although the quantitative information is optional. Also, the model is easy to grasp for small specifications but hard to understand for large specifications.

1.1.2 CHIMP

Another useful constraint-based model is CHIMP [CPS96] [CPS98]. In CHIMP, temporal synchronization requirements are represented as sets of difference constraints of the form: x – y <= a, where a is a rational number, and x and y are variables representing the starting or ending time of the media objects in the temporal specification.

Difference constraints are further associated with a weighted graph G = (V, E). V contains the variables appearing in difference constraints. Each graph contains two vertices Vs and Ve that represent the starting and ending time of the entire specification respectively. For each difference constraint x – y <= a, the graph contains an arc from the node representing variable x to the node representing variable y, whose weight is a.

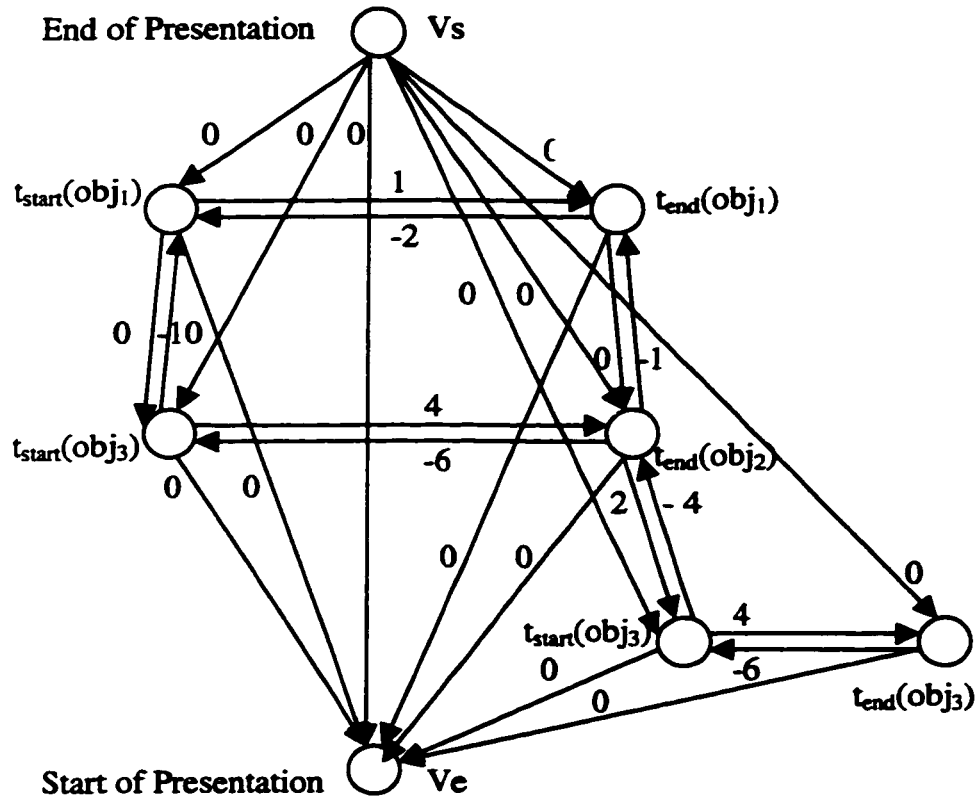


FIGURE 2: An Example of CHIMP

An example of CHIMP is shown in FIGURE 2 for the same presentation used in FIGURE 1 (Firefly). Note that the temporal constraints are different from Firefly. In CHIMP, more complicated constraints can be expressed. For instance, the constraints between $t_{start}(obj_1)$ and $t_{start}(obj_2)$ requires that $t_{start}(obj_2)$ should occur after 10 seconds $t_{start}(obj_1)$ happens. Obviously, CHIMP can express both the qualitative temporal relationships and the quantitative relationships among media objects. In particular, CHIMP satisfies more quantitative requirements than Firefly because any temporal relationship in it is expressed by arbitrary difference constraints. Also, the model is very easy to grasp for small specifications but hard to understand for large specifications.

1.1.3 Interval Constraint Graph Model (ICGM)

Because constraint-based models, in particular difference constraints, have a more flexible and stronger ability to express both qualitative and quantitative temporal relationships among media objects, in this thesis we propose and adopt a difference constraint-based model called interval constraint graph model (ICGM). ICGM can provide flexible, constraint-based specifications for specifying the temporal aspect of multimedia presentations.

ICGM consists of nodes and edges where the nodes represent events, an edge with label [min, max] represents a temporal interval of durations of events from min to max. An interval constraint graph describes the temporal ordering of media objects by capturing information about their logical and quantitative temporal dependencies. The

range of possible durations of a dynamic object to be presented is modeled by an edge labeled by its appropriate interval. In fact, the expression from ICGM can be translated into difference constraints. For example, if an edge with a label [a, b] connects node/event X to node/event Y, and X, Y occur at time x, y respectively, they can be expressed by two difference constraints: $y - x <= b$ and $x - y <= -a$. Hence, ICGM is a difference constraint-based model.

FIGURE 3 is an example of ICGM, which models a multimedia presentation similar with the example for CHIMP and Firefly. In this example, $t_{start}(obj_2)$ should occur in the time interval [0,10] after $t_{start}(obj_1)$. Similarly the rest of the temporal requirements are specified by the respective time intervals.

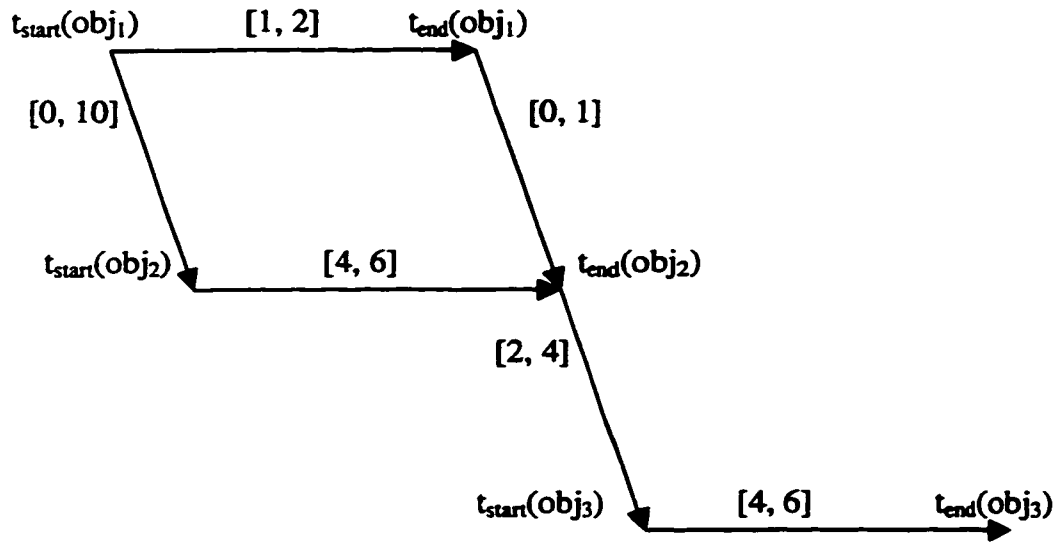


FIGURE 3: An Example of ICGM

Like Firefly and CHIMP, the ICGM is very easy to grasp for small specifications, but hard to understand for large specifications.

1.2 Presentation Schedule and QoS Consideration

1.2.1 Presentation Schedule

A temporal synchronization model can specify the temporal relationships among media objects in a multimedia presentation. Furthermore, in order to present a multimedia document including continuous media, such as video and audio, the presentation system needs to know when to present the media objects to the user or how to schedule the presentation. In other words, at run time, the presentation of multimedia objects has to be carried out in accordance with the temporal constraints between the media objects. This synchronization mechanism can be reflected by a presentation schedule produced from the temporal synchronization specification.

A presentation schedule describes the starting times and durations of the presentation of media objects in a multimedia document, and satisfies the flexible temporal constraints associated with that multimedia document.

Given a set of temporal constraints or temporal synchronization specifications, there does not always exist a schedule satisfying this set of constraints. A schedule is the assignment of a point in a time line to every temporal entity or event such that all temporal constraints are satisfied by this assignment. A schedule can only be produced by a consistent temporal specification.

Many researchers have developed algorithms to schedule presentations. The Firefly system uses the simplex algorithm to solve for a presentation schedule. Also, Firefly allows users to specify an optimal duration for a multimedia object along with

costs for shrinking and stretching the duration. However, they need to pay more attention to the determination of the cost for each shrinking and stretching. K.S.Candan et al. use much more efficient graph algorithms [CPS98] for the presentation schedule, but they are not concerned with the optimal schedule. In a concurrent related work, Kim and Song [KS95] use a constraint-based approach to provide an elastic time model that allows the duration of the objects to shrink and stretch as necessary. They use additional constraints to keep the duration of the objects at their optimal values. They also use linear programming to solve for a presentation.

1.2.2 QoS Consideration

It has been recognized that an effective support for multimedia applications must provide Quality of Service (QoS) guarantees. Basically from the perspective of users, the consideration of QoS involves two categories of qualities for a multimedia presentation.

First, users need to specify media qualities, such as the type of media, the resolution of images, the number of frames per second for a video object, etc. The media qualities that a user has chosen are also related to service charges. Generally speaking, the higher the required quality, the higher is the charge. This policy will discourage users from always using the highest quality. Users can directly define the parameters of this category of qualities.

Second, users need to specify the qualities of temporal constraints. For the purpose of temporal synchronization, we may assign an interval constraint for two events in a specification. Furthermore, a presentation schedule can be produced according to the

temporal specification. The extent of satisfying these constraints in the schedule can be reflected by the qualities of constraints. Users can define the qualities of temporal constraints by assigning some QoS functions for each temporal constraint of the interval constraint graph. The QoS functions take the duration between events as variables and output values to measure the extent of satisfying the temporal constraints. In this way, these QoS functions can express the importance of a temporal constraint or the user's preference in the presentation. For instance, the temporal constraint between the start and end of a video object is more important than that between the start and end of a text object; the importance can be reflected by different coefficients of QoS functions.

An example of QoS functions corresponding to FIGURE 3 is shown in FIGURE 4.

QoS function for the constraint between $t_{start}(obj_1)$ and $t_{end}(obj_1)$: x - 1 and 2 - x
QoS function for the constraint between $t_{start}(obj_2)$ and $t_{end}(obj_2)$: x - 4 and 6 - x
QoS function for the constraint between $t_{start}(obj_3)$ and $t_{end}(obj_3)$: x - 4 and 4 - x
QoS function for the constraint between $t_{start}(obj_1)$ and $t_{start}(obj_2)$: x - 0 and 10 - x
QoS function for the constraint between $t_{end}(obj_1)$ and $t_{end}(obj_2)$: x - 0 and 10 - x
QoS function for the constraint between $t_{end}(obj_2)$ and $t_{start}(obj_3)$: x - 2 and 4 - x

FIGURE 4: An Example of QoS Model

Since there can be many feasible schedules for a given ICGM or temporal specification, the problem is which one should be chosen as a presentation schedule. From the user's perspective, the schedule with maximum cumulative QoS count should be the best, so we adopt a linear programming (LP) solver which can produce an optimal schedule with the maximum QoS count to derive the schedule. In this way, our flexible temporal model can help to derive the optimal presentation schedule in the sense of QoS requirements of temporal constraints.

1.3 Delivery Schedule

In a distributed environment or a client/server system, the media objects composing a document may be dispersed over a computer network. These objects have to be retrieved or delivered from their storage sites (servers), transferred to the presenting system (client) and presented to users by the client. To complete a multimedia presentation, servers need to know when to deliver media objects. That is to say, the delivery of media objects must be on schedule. Therefore, a communication synchronization mechanism is needed in a distributed environment, and a delivery schedule can be used for this purpose. The delivery of multimedia objects from the media servers is influenced by factors such as temporal constraints of the presentation, bandwidth or throughput offered by the network service provider, and the buffer resources on the client system.

The buffer constraints and bandwidth/throughput constraints are system-dependent. The buffer resources are dependent on their availability in the client system. The available throughput can vary depending on the type of network and the load on the network.

A delivery schedule specifies the time instants at which servers should deliver media objects in multimedia documents to the client system.

Some researchers have studied delivery or retrieval schedules. The latter specifies the time to request delivery of objects from clients and hence has the same meaning as delivery schedules. In [LG91], presentation schedules are based on petri net description

of temporal specifications, but they are fixed before the generation of retrieval schedules. Retrieval schedules are derived by assuming certain throughput to be provided by the network. Moreover, estimates for the buffer resource requirements on the client system are made based on the derived retrieval schedule and the assumed network throughput. In [RPT94], the authors use petri net model to describe temporal specifications, and they produce retrieval schedules on fixed presentation schedules. In [GL96], Gibbon and Little give an effective way to create a data retrieval schedule based on delay estimation and using real-time scheduling. They try to guarantee that media objects arrive at the client system before their presentation time, but not so early as to overflow the allocated buffer space of the client system. In addition, the estimation model does not assume that all packets must be transmitted on the same physical path through the network, so it can be used for a wide type of networks.

Delivery schedules should be in concordance with the transmission ability of the network system. Otherwise, too much delay will occur in the multimedia presentation, or the client system needs a very large buffer to store the media objects. To avoid both underflow and overflow of the buffer in the client system, we derive the delivery schedule based on the estimated values of system traffic in client, network, and server and other parameters, after the system produces an optimal presentation schedule.

However, because of the dynamic behavior of system traffic, there are no guarantees that the media object can arrive at the destination on schedule. Therefore, we should have some kind of remedy in case of occurrence of delays. We hope to design a distributed scheduling to allow fine-tuning at both server and client sites to adapt to dynamic changes of system traffic and solve the problem by means of rescheduling.

1.4 Run Time Rescheduling

The static delivery schedule and presentation schedule are not sufficient to maintain a simultaneous object delivery and presentation since the multimedia system introduces random disturbances at run time in the client system because of network delay or server delay. Random network delay or server delay destroys the continuity of the data stream by introducing gaps and jitters during the data transmission. Delay jitters can be removed with a FIFO buffer at the destination. However, if synchronization errors caused by transmission gaps cannot be tolerated by human perception, i.e., scheduling or predicting traffic is not sufficient to maintain a synchronized data delivery, certain compensations at the client system are necessary.

In [RVR92], Rangan et al. present an inter-media synchronization technique. In this case, a centralized multimedia server uses feedback transmitted by the client system to detect the inconsistencies among media objects. The correction of inconsistencies is made at the multimedia servers by speeding up or slowing down the traffic on the media data transmission. A real time synchronization method is needed to display a document in real time and to preserve the synchronization. It might be difficult for this method to detect and correct inconsistencies before the user can notice them when the network traffic is heavy.

In [LLG94], the compensation mechanism is called stream synchronization protocol (SSP) that allows for synchronization recovery to preserve a high quality multimedia display at the client system. SSP uses Synchronization Quality of Service

parameters to guarantee simultaneous delivery of different types of data streams. Lamont et al. use the concept of an "intentional delay" to adjust their compensation times to recover from network fluctuations without having to discard or skip late packets. This method is very well suited for applications such as multimedia news on-demand services. However, unexpected inconsistencies still happen for some periods because the method just schedules the traffic and does not predict the events causing inconsistencies. Therefore, it is not suitable for multimedia applications with strict requirements of temporal relationships among media objects.

Our approach is to use a flexible temporal constraint specification for deriving an optimal presentation schedule. Hence, if inconsistencies happen in the client system, we can choose another schedule rather than the optimal schedule such that the multimedia presentation can keep consistent temporal constraints among media objects. In order to derive another feasible schedule, we adopt the method of rescheduling based on the retiming method described latter. As soon as the client system receives the signal that a scheduled event cannot occur on time, it can reschedule the presentation to satisfy the temporal requirements. Because several kinds of events for different situations can trigger rescheduling, our rescheduling strategy is well suited for a distributed environment. For example, it can adapt to dynamic changes of system traffic including network traffic and server traffic. In addition, it allows rescheduling at both server and client sites.

In summary, at run time the system needs to detect or predict synchronization errors or delays, and reschedule the presentation to eliminate or minimize the inconsistencies caused by the delays to satisfy the temporal requirement of a multimedia presentation.

To trigger rescheduling of future events in the client system, the system detects, at time t, whether the scheduled event at some future time t' > t can occur on time or not because of network delay or server delay. If the scheduled future event cannot occur on time, all events after time t should be rescheduled. The rescheduling is based on the rescheduling policy, i.e., the retiming method, plus the knowledge of some delays (known from delay estimations) needed by the originally scheduled event to be rescheduled for time t'.

In this chapter, we have introduced the basic problem and relevant concepts in the area of distributed multimedia document presentation, in particular distributed multimedia synchronization. The rest of the thesis is organized as follows. CHAPTER 2 discusses some requirements related to multimedia synchronization in a distributed environment. They are important in order to understand the design of our synchronization scheduler. After introducing the temporal constraint graph model and associated QoS model, CHAPTER 3 develops how to produce an optimal static schedule based on the model using a linear programming solver. CHAPTER 4 describes the rescheduling strategy including detailed algorithms. Also, it gives some examples. The aim of CHAPTER 5 is to detail our design of a scheduler for a distributed multimedia presentation system. It also outlines the architecture designed for the distributed multimedia presentation system and describes the important parts in brief. CHAPTER 6 describes some important details of implementation in a simulated distributed environment. It also analyzes some simulation experiments. CHAPTER 7 draws detailed conclusions and also contains proposals for further work.

CHAPTER 2 Requirements of Multimedia Document Presentation in a Distributed Environment

2.1 Distributed Environments

Distributed environments are composed of processor nodes connected by a network to allow communication and data sharing. Processors manage the scheduling of processes and communication between machines. Users can share system resources such as storage services. With the rapid increases in processor and network speeds, distributed environments are capable of handling continuous media such as digital audio and video.

In distributed environments, workstations for multimedia users have been augmented with speakers, cameras, microphones and other electronic equipment for the input and output of audio and video data. Storage services and networks have been extended to support multimedia data. Display of multimedia items in a multimedia document is achieved by transporting the data from its sources into the user's workstation.

2.2 End to End Communication

A distributed multimedia system should support retrieval, processing, and transmission of both static and time-dependent continuous media. That requires a very strict network traffic requirement, in terms of bandwidth, delay and delay jitter. End to

end refers to the entire communications system, from the data producers or sources (e.g. storage) to the data consumers or sinks (e.g. displays).

The transmission rate (bandwidth) at the transmitter should match the consumption rate at the receiver. If the average arrival rate is greater than the consumption rate, either an infinite buffer is required or a playout of the stream can only be sustained for a limited time (determined by the buffer size). On the other hand, if the average arrival rate is smaller than the consumption rate, the initial delay is very long; consequently, the system response time becomes long. Therefore, end to end transmission bandwidth should be at least equal to (or close to) the consumption rate.

End to end delay is the sum of all delays in all the components of a multimedia system, including disk access, ADC (analog to digital conversion), encoding, host processing, network access, network transmission, buffering, decoding, and DAC (digital to analog conversion). Furthermore, We can distinguish them as three parts. The first part is the server access delay which is the sum of delays in disk access, ADC, encoding, and host processing. For orchestrated multimedia presentations, there are no ADC or encoding; therefore the server access delay is the sum of delays in disk access and host processing. The second part is the network delay which is the sum of delays in network access and the network transmission. The third part is the client process delay which is the sum of delays in buffering, decoding, and DAC. Then, in order to present a media object in a distributed environment, the total end to end delay is given by the sum of the delays of these three parts.

Delay jitter should be removed to guarantee end to end performance. Continuous media must be sampled and played at regular intervals, or their perceivable qualities will

be unacceptably low. When using a packet switched network, data packets do not arrive at the destination in fixed intervals as required by continuous media because of variations in processing and transmission time. This delay variation is called delay jitter. Delay jitter can be removed with a first in first out (FIFO) buffer at the destination. In addition, in order to meet the multimedia communication, the buffer should not overflow or underflow. Moreover, the buffer size should not be too large; a large buffer means that the system is expensive and the overall end to end delay is large.

2.3 Real Time Synchronization

There are two basic approaches for multimedia communication synchronization in a distributed environment. One is the real-time synchronization scheme, and the other is the store-and-forward scheme. In the latter approach, the multimedia components of a document are assembled and buffered at the client system before they are played back according to the presentation schedule. This approach is simple to apply, but it may require huge buffer storage and cause larger response delay for the user interaction. Therefore, in order to support real-time multimedia applications, a real-time synchronization approach is useful and will be adopted in this thesis. Instead of delivering all of the media objects before the presentation, we want to deliver the media objects only when they are needed. In this way, the presentation and object deliveries may overlap thereby reducing both the storage requirement and the response time.

The major disadvantage of overlapped deliveries is that, the multimedia presentation can be affected by object deliveries, and some delivery decisions must be

taken during run time. For instance, if an object is not delivered at the right time because of server access delay or network delay, the multimedia presentation and delivery have to be changed and must be rescheduled at run time such that the delayed object will not affect the quality of the presentation.

2.4 Multimedia Document Presentation in a Real Time Synchronization System

2.4.1 Multimedia Objects

A multimedia document is composed of media objects that are to be presented at different time instants and for different durations. Based on the way the objects are to be delivered from the servers, we classify them as:

- Atomic Objects: These objects need to be received as a whole at the client system before the presentation can start. For example, still-image files are atomic objects.
- Stream Objects: These objects can be presented to the viewer as soon as some portion of them is received at the client system. The rest of the object is then continuously fed to the viewer. For example, video and audio objects can be considered as stream objects in a real-time synchronization system; but they must be considered as atomic objects in a store-and-forward system.

Thus atomic objects must be present in the buffers of the client system entirely before their presentations. Stream objects do not need to be delivered as a whole before their

presentations. However, in order to reduce delay jitter and to smooth their display, such objects usually require some fraction to arrive at the client system before their presentations. They can be consumed from the buffers during the presentation. Note that, in order to prevent the underflow and overflow of the buffer, the consumption rate of an object must match the average delivery rate (throughput) of that object. Hence, if the network cannot provide the required throughput matching with the consumption rate, we can reduce the throughput requirement by buffering a larger portion of the object at the client system (to match the throughput and consumption rate).

2.4.2 Object Delivery and Presentation in a Multimedia Presentation

In order to present an atomic object, say obj, after the server access delay ds(obj), we transmit the object through the network to clients in network delay dn(obj) which is the duration from the time instant at which a sever starts to deliver the object obj onto communication channels until the time instant at which a client receives the whole object. Before the client system can present the object, a client process delay dc(obj) is needed. Then, we can derive a delivery schedule by computing these delays. We analyze them as follows.

(1) In an orchestrated multimedia presentation system, media objects are retrieved from storage devices. The access time and the transfer rate of storage devices are determined by the characteristics of multimedia servers. In other words, it is server dependent.

(2) Obviously, the client process delay is client system dependent; therefore, it can be computed by the local client system and transferred to the local scheduling system. For instance, the client process delay with decoding and without decoding are computed differently.

(3) The network delay includes the network access delay and network transmission delay.

The network propagation delay or transmission delay can be seen as a constant for many types of networks. For instance, the network propagation delay or transmission delay is often assumed as 100μs for ISDN.

In the case of a stream object, a chunk of frames is to be delivered before the start of a presentation. The size of the chunk depends on the type of the media, delay jitter requirements, and the display hardware at the client system, so it is often determined by the client system.

In this chapter, we discussed some important requirements related to multimedia synchronization. The next chapter deals with the derivation of an optimal presentation schedule.

CHAPTER 3 Optimal Static Presentation Schedule

In this chapter, we present an approach to solve the problem of obtaining an optimal schedule for a given temporal specification. We first formulate our interval constraint graph model. Then, we generate a set of linear forms from the given temporal specification. The linear form constraints then can be solved by a linear programming (LP) solver in polynomial time [NS96].

3.1 Interval Constraint Graph Model (ICGM)

In general, a multimedia document is composed of a set of media objects, along with an associated set of presentation requirements. These requirements could include temporal and spatial constraints that have to be satisfied during a presentation. The thesis focuses on the satisfaction of temporal requirements, i.e. temporal synchronization. In addition, temporal synchronization is distinguished into two types: intra-stream and inter-stream. The intra-stream synchronization is concerned with delivering each media object in time to meet the respective playout deadline, while the inter-stream synchronization is required to describe temporal relationships among media objects. We are interested in the latter and assume that the former is always satisfied. In other words, our model does not support the intra-stream synchronization in an object. Moreover, the system has to guarantee Quality of Service (QoS) for presenting the multimedia document.

In order to model the above requirements, we develop the Interval Constraint Graph Model (ICGM). The ICGM consists of two related parts: the Interval Constraint

Graph and the QoS Model to describe temporal constraints among media objects and QoS requirements respectively.

3.1.1 Constraint Graph Models

3.1.1.1 Interval Constraint Graph (ICG)

Our Interval Constraint Graph is based on temporal constraint networks that were first developed by R.Dechter et al. [DMP91] and have been proved powerful and efficient. It takes into account both the logical parts of a multimedia document and its quantitative aspect of temporal requirements.

The temporal requirements of a multimedia presentation are captured by a labeled directed acyclic graph G = (N, A, T) where N represents a set of event (start/end event) nodes, A represents a set of edges and T represents the timing constraints separating the events so connected by the edges. In particular, an edge (e_i, e_j) in A is labeled with (t_{ij}.min, t_{ij}.max) in T. Semantically, it represents the requirement that e_j must happen no earlier than t_{ij}.min and no later than t_{ij}.max after e_i has happened. Such a graph is called Interval Constraint Graph (ICG).

An example of ICG can be seen in FIGURE 3.

3.1.1.2 Consistency of ICG

An ICG may be inconsistent (unscheduleable), i.e. there does not exist a schedule that can satisfy all of the requirements specified by the ICG. For example, the ICG in FIGURE 3 is inconsistent. To check this, it is sufficient to observe that the maximal separation time between $t_{start}(obj_1)$ and $t_{end}(obj_2)$ via the constraints $t_{start}(obj_1) \rightarrow t_{end}(obj_1) \rightarrow t_{end}(obj_2)$ is 2 + 1 = 3. However, the minimal separation time between the same pair of nodes via the constraints $t_{start}(obj_1) \rightarrow t_{start}(obj_2) \rightarrow t_{end}(obj_2)$ is 0 + 4 = 4. Then these two conditions can never be consistently satisfied. Hence the notion of consistency is born. In short, whether an ICG can be satisfied consistently is the consistency of the ICG.

3.1.1.3 Temporal Constraint Graph (TCG)

To formalize the notion of consistency, it is useful to transform an ICG into an equivalent temporal constraint graph (TCG). The transformation involves converting an edge (e_i, e_j) in an ICG into two edges $\{(e_i, e_j), (e_j, e_i)\}$ in the corresponding TCG and labeling (e_i, e_j) with t_{ij}.min and (e_j, e_i) with $-t_{ij}$.max.

The semantics of TCG can be defined by the following inequality. Suppose t_i is the time of occurrence of e_i, and t_{ij} is the label on edge (e_i, e_j). Then the following axiom holds:

$$t_j - t_i <= t_{ij} \qquad \text{(A0)}$$

We could use A0 to verify the equivalence of ICG and TCG. In particular, given (e_i, e_j) labeled with t_{ij} and (e_j, e_i) labeled with t_{ji}. Then from A0 and (e_i, e_j), we deduce that e_j can happen no later than t_{ij} time units after e_i has happened. In addition, from A0

and (e_j, e_i), we could also deduce that e_i can happen no later than t_{ji} time units after e_j, or equivalently, no earlier than $-t_{ji}$ time units before e_j. Hence the equivalence with (e_i, e_j) labeled with $(t_{ij}, -t_{ji})$ in the ICG.

3.1.1.4 Consistency of TCG

The consistency of a TCG can be easily checked using the following theorem.

Theorem 3.1: A TCG is consistent iff it does not contain strictly positive cycles.

Proof: Suppose there is a strictly positive cycle, C, consisting of nodes $e_1, e_2, \ldots, e_k = e_1$ with the cycle length $L > 0$. Summing the inequalities in A0 along C, yields $t_1 - t_1 <= L$.

From the definition of t_{ij}, $t_1 - t_1 <= t_{11} = L$ means that e_1 should happen no later than L time units after e_1 has happened, which is a conflict and cannot be satisfied. Conversely, if there is no positive cycle in the TCG, then the longest path between each pair of nodes is well defined. For any pair of nodes, the longest paths satisfy all constraints in the TCG, hence are schedules of the given TCG.

QED

As a simple example, the ICG of FIGURE 3 is transformed into its TCG counterpart in FIGURE 5. Consequently, the directed cycle: $t_{start}(obj_1) \rightarrow t_{start}(obj_2) \rightarrow t_{end}(obj_2) \rightarrow t_{end}(obj_1) \rightarrow t_{start}(obj_1)$ has a cycle of length 1 and hence the TCG is inconsistent.

The consistency of a TCG can be checked by verifying if the graph contains strictly positive cycles, a problem that is solvable in polynomial time.

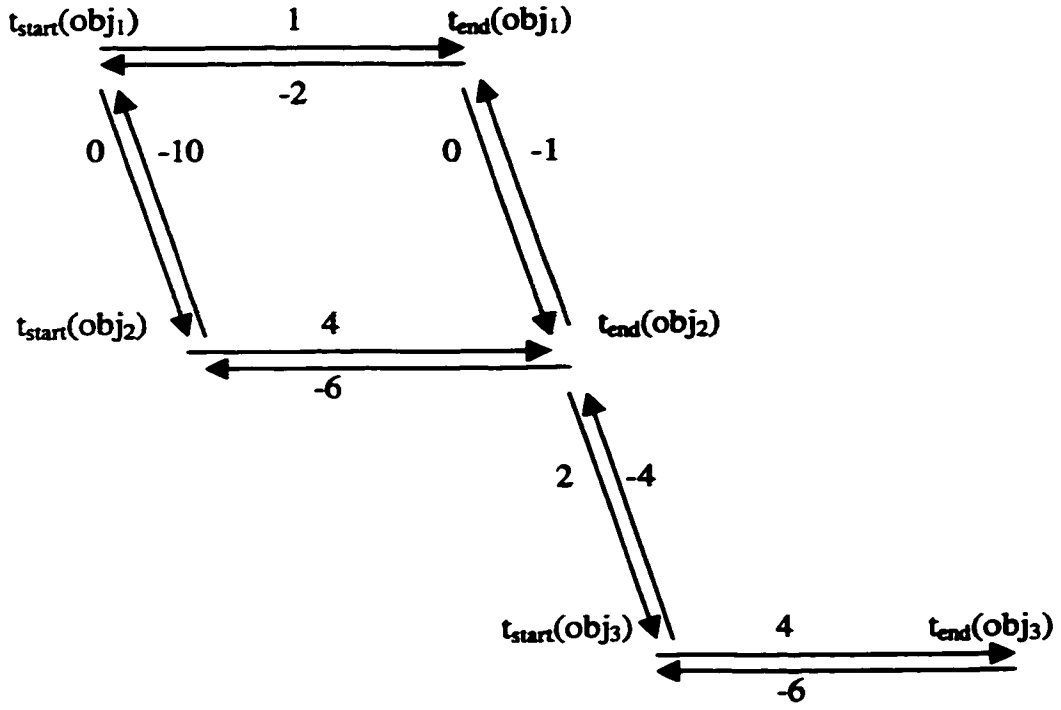


FIGURE 5: The TCG Corresponding to the ICG in FIGURE 3

3.1.1.5 Feasible Schedule

Users can determine temporal constraints in an ICG or a TCG with the minimum and maximum values of durations. That means that there may exist a set of solutions satisfying the temporal constraints, and each of them is a feasible solution (schedule).

A feasible schedule is an assignment of values to $(t_1, t_2, \ldots t_n)$, where t_i, $1 <= i <= n$, is the time of occurrence of e_i, such that they satisfy all the constraints specified in the TCG via A0. As an example, suppose the TCG of FIGURE 5 is modified to become that in FIGURE 6 in which the constraints between $t_{start}(obj_2)$ and $t_{end}(obj_2)$ have been

changed from [4, 6] to [3, 6]. Then the schedule indicated in FIGURE 6 is feasible. It is easy to verify all the constraints are satisfied by the schedule.

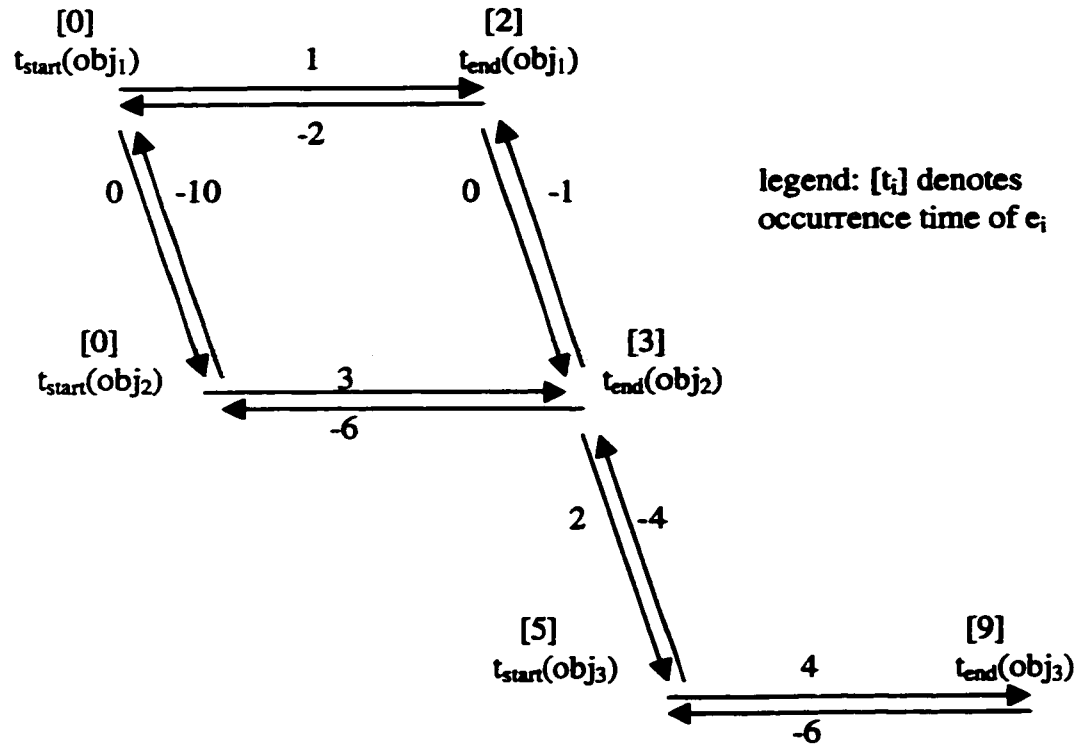


FIGURE 6: A Feasible Schedule in a Constraint TCG

3.1.2 Quality of Service Model

Assigning Quality of Service (QoS) to a multimedia presentation and guaranteeing them is an effective support for the multimedia presentation. The necessary QoS requirements depend on the media and presentations. In our model, users can specify the qualities of temporal constraints for a multimedia presentation, so each presentation has its own Quality of Service. Hence, in addition to satisfying temporal constraints in a graph, a schedule needs to meet the constraints defined in the QoS model as the guarantee for the QoS.

3.1.2.1 QoS function

Not all presentations are of the same quality. Here we are concerned about the qualities of temporal constraints, i.e., how to keep a presentation with consistent temporal constraints. To model the temporal constraint qualities of a presentation, and hence a schedule, we attach a quality function on each interval constraint of the ICG. In particular, given (t_{ij}.min, t_{ij}.max), the choice of t_{ij} within the interval returns a quality factor. For example, the presentation of a video clip should last between 20 and 25 seconds. An optimal presentation might be for 23 seconds. A rough quality of service function can be applied to the duration of the video clip as scheduled, such as shown in FIGURE 7. Also, we could model the separation time between the end of an audio clip and the start of a new text display with a similar quality function.

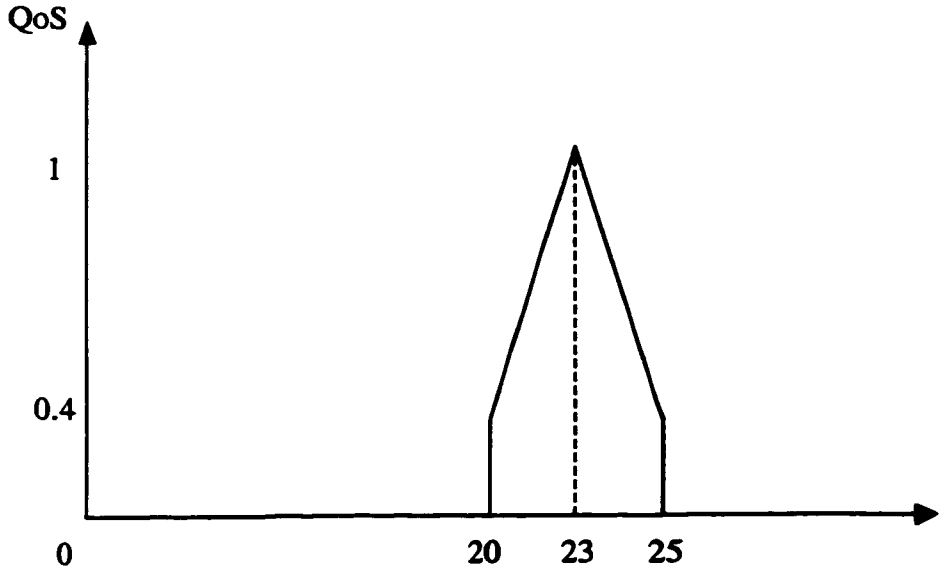


FIGURE 7: A Quality of Service (QoS) Function

We take a simple form of quality of service function consisting of a triangular function as shown in FIGURE 8. Intrinsically it is assumed that there is an optimal value for each temporal separation $d_{ij} = (t_j - t_i)$, and deviations from this optimal value (d_{ij}*) will lead to degradation of quality modeled by a linear quality function of the form a x $(d_{ij} - d_{ij}^*) + b$.

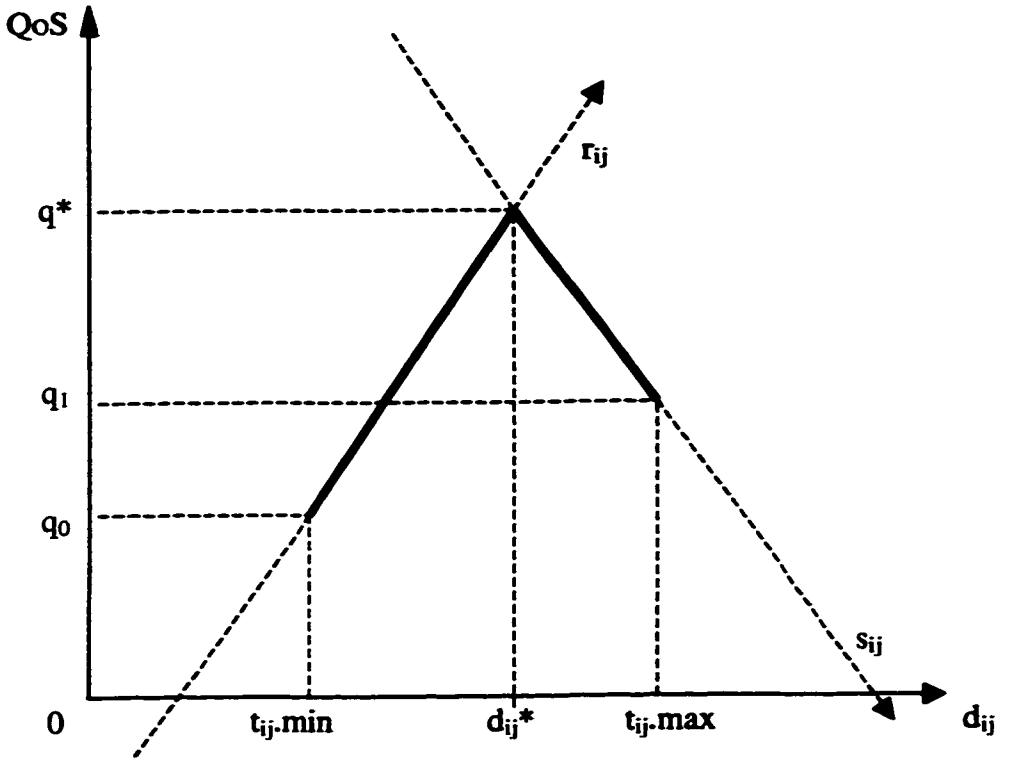


FIGURE 8: General QoS Function

The QoS functions have the linear form: a x + b, where a and b are coefficients. Users can assign different coefficients to express the importance of a temporal constraint or the user's preference in the presentation. For instance, the temporal constraint between the start and end of a video object is more important than that between the start and end

of a text object. Hence, with a constraint interval [10, 12] of durations between the start and end of an object, users may define the QoS functions as x-10 and 12- x for a text object, whereas users may define the QoS functions as 3(x-10) and 3(12-x) for a video object. When producing the optimal schedule, the LP solver will try to achieve a better presentation of the video object rather than the text object. In this way, the importance can be reflected by different coefficients.

Our QoS model is an approximate model, and the remaining parts of the thesis are based on this model.

3.1.2.2 Optimal scheduling

Generally speaking, users hope to present a document achieving the best performance or quality. Given a TCG and the associated quality of service functions, an optimal scheduling problem can be defined to derive a feasible schedule that has the best quality of service. Here "the best" means that a multimedia document can be presented in the best qualities of temporal constraints, or that leads to the best presentation. In the next section, we will formulate the problem as a linear programming problem.

3.2 Production of an Optimal Static Schedule

TCG Constraints

Each TCG constraint of the form: $t_j - t_i <= t_{ij}$ is a linear constraint and is included in the input constraints set.

QoS Constraints

Referring to FIGURE 8, the quality given by a chosen (t_i, t_j) under the interval constraint (t_{ij}.min, t_{ij}.max) is specified by:

- $d_{ij} = t_j - t_i$
- $r_{ij} = q_0 + a \times (d_{ij} - t_{ij}.min)$
- $s_{ij} = q_1 + b \times (t_{ij}.max - d_{ij})$
- $q_{ij} <= r_{ij}$
- $q_{ij} <= s_{ij}$

Note that r_{ij} models the part of the QoS function linearly increasing from t_{ij}.min with a value of q_0 to d_{ij}* with a value of qmax(q*). Extrapolation of this function is allowed to the full (-infinity, infinity) range. Similarly, s_{ij} models the part of the QoS function linearly decreasing from d_{ij}* with a value of qmax(q*) to t_{ij}.max with a value of q_1. Again we allow this function to be extrapolated to the full range of real numbers. Finally, the actual value of the quality of service is defined by min(r_{ij}, s_{ij}) within the feasible range dictated by the TCG constraints. This is the crucial trick in our work so that the problem can be formulated as a simple linear programming problem solvable in polynomial time. Unfortunately, for arbitrary QoS functions, the problem becomes non-linear and will need more complex solutions.

Objective Function

The objective function to be optimized is given by:

Maximize $\sum_{ij} q_{ij}$

The range of i and j traverses all constraints in the specification. For those constraints that are not defined with QoS functions, we assume the corresponding q_{ij} is zero.

Solution and Complexity

The linear programming problem so formed can be solved in polynomial time. The Ellipsoid Method requires at most O((number of constraints * number of variables^3 + number of constraints^4) * the length of input data for the system) arithmetic operations [NS96].

Example

We take the TCG in FIGURE 9 as an example to show how to produce an optimal schedule.

The temporal constraints defined in the TCG are as follows:

(1) $t_{start}(obj_1) - t_{end}(obj_1) <= -3$

(2) $t_{end}(obj_1) - t_{start}(obj_1) <= 1$

(3) $t_{start}(obj_2) - t_{end}(obj_2) <= -7$

(4) $t_{end}(obj_2) - t_{start}(obj_2) <= 3$

(5) $t_{start}(obj_3) - t_{end}(obj_3) <= -7$

(6) $t_{end}(obj_3) - t_{start}(obj_3) <= 3$

(7) $t_{start}(obj_1) - t_{start}(obj_2) <= -10$

(8) $t_{start}(obj_2) - t_{start}(obj_1) <= 0$

(9) $t_{end}(obj_1) - t_{end}(obj_2) <= -3$

(10) $t_{end}(obj_2) - t_{end}(obj_1) <= 0$

(11) $t_{end}(obj_2) - t_{start}(obj_3) <= -4$

(12) $t_{start}(obj_3) - t_{end}(obj_2) <= 2$

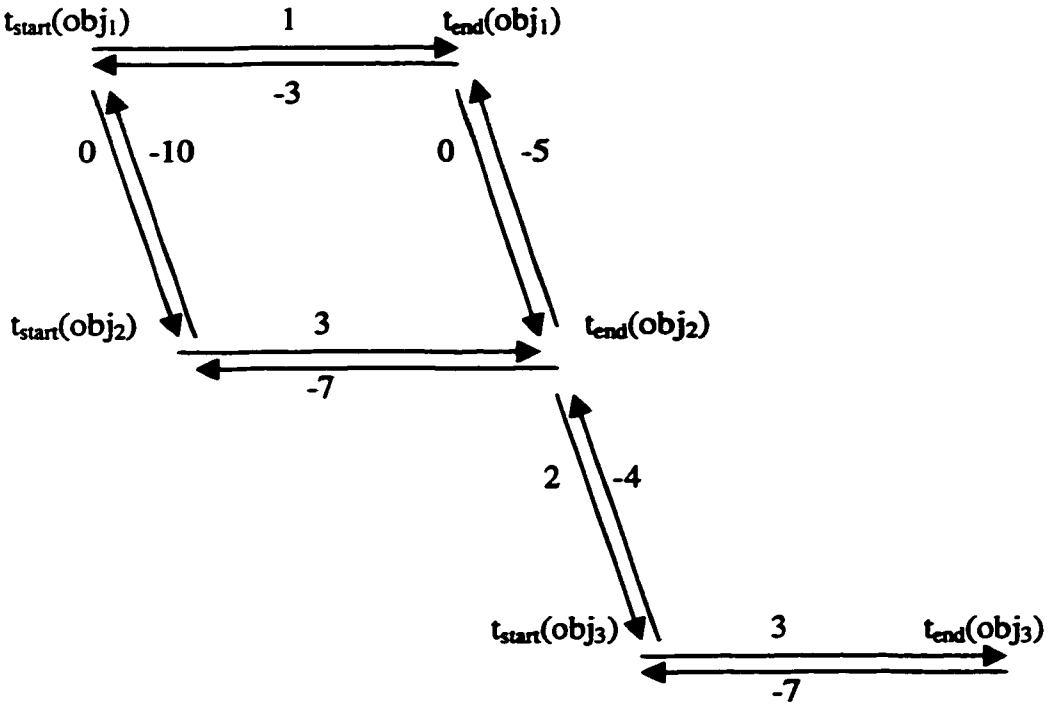


FIGURE 9: A Consistent TCG

Suppose we define the QoS functions for the TCG as:

For the constraints between $t_{start}(obj_1)$ and $t_{end}(obj_1)$, the functions are 3x-3 and 9-3x.

For the constraints between $t_{start}(obj_2)$ and $t_{end}(obj_2)$, the functions are 3x-9 and 21-3x.

For the constraints between $t_{start}(obj_3)$ and $t_{end}(obj_3)$, the functions are 3x-9 and 21-3x.

For the constraints between $t_{end}(obj_2)$ and $t_{start}(obj_3)$, the functions are x and 4-x.

Then, we have:

(13) $q_{sle1} <= 3*(t_{end}(obj_1) - t_{start}(obj_1)) -3$

(14) $q_{s1e1} <= 9-3*(t_{end}(obj_1) - t_{start}(obj_1))$

(15) $q_{s2e2} <= 3*(t_{end}(obj_2) - t_{start}(obj_2)) -9$

(16) $q_{s2e2} <= 21-3*(t_{end}(obj_2) - t_{start}(obj_2))$

(17) $q_{s3e3} <= 3*(t_{end}(obj_3) - t_{start}(obj_3)) -9$

(18) $q_{s3e3} <= 21-3*(t_{end}(obj_3) - t_{start}(obj_3))$

(19) $q_{e2s3} <= (t_{start}(obj_3) - t_{end}(obj_2))$

(20) $q_{e2s3} <= 4 -(t_{start}(obj_3) - t_{end}(obj_2))$

Thus we have got 20 linear form constraints from (1) to (20) with 10 variables, i.e., $t_{start}(obj_1)$, $t_{end}(obj_1)$, $t_{start}(obj_2)$, $t_{end}(obj_2)$, $t_{start}(obj_3)$, $t_{end}(obj_3)$, q_{s1e1}, q_{s2e2}, q_{s3e3} and q_{e2s3}.

The objective function is $(q_{s1e1}+ q_{s2e2} + q_{s3e3} + q_{e2s3})$.

The LP solver can solve the constraints and get the following results:

(21) $t_{start}(obj_1) = 0$

(22) $t_{end}(obj_1) = 2$

(23) $t_{start}(obj_2) =0$

(24) $t_{end}(obj_2) = 5$

(25) $t_{start}(obj_3) = 7$

(26) $t_{end}(obj_3) = 12$

(27) $q_{s1e1} = 3$

(28) $q_{s2e2} = 6$

(29) $q_{s3e3} = 6$

(30) $q_{e2s3} = 2$

Note that (21) to (26) consist of a feasible schedule shown in FIGURE 10 for the TCG in FIGURE 9, and from (27) to (30), the corresponding maximum objective value $(q_{s1e1} + q_{s2e2} + q_{s3e3} + q_{e2s3}) = 17$ can be obtained. Thus the feasible schedule is an optimal schedule in the sense of QoS for the TCG because it holds the maximum QoS value. In other words, the multimedia presentation according to the optimal schedule can achieve the best performance or QoS.

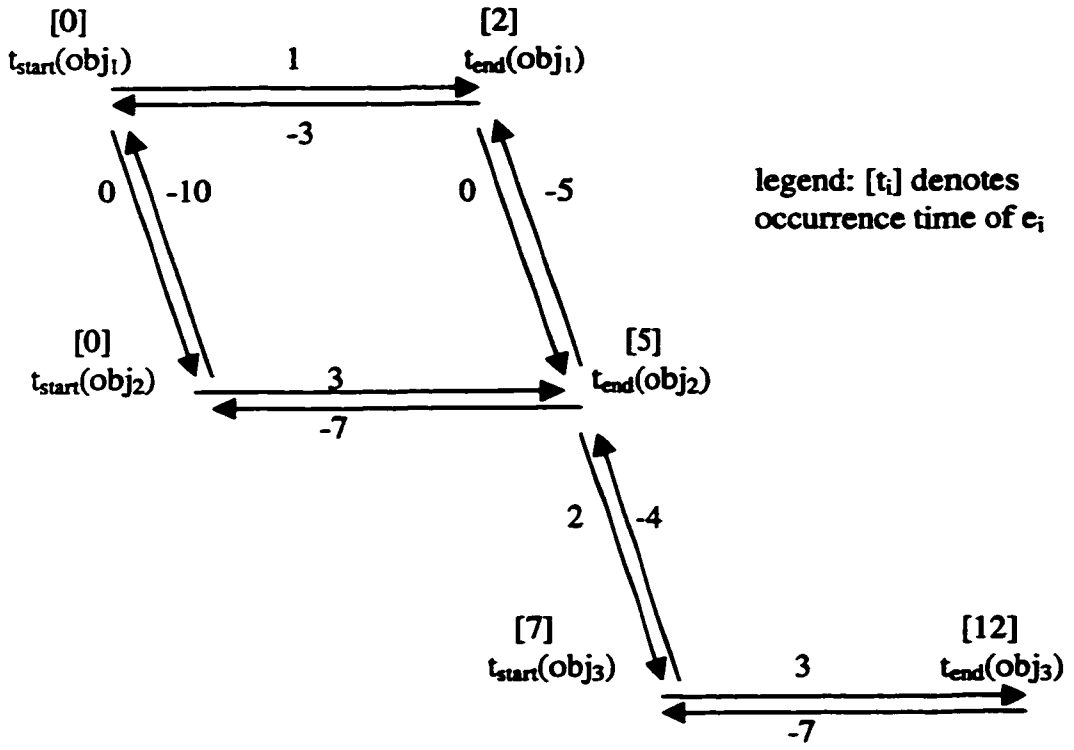


FIGURE 10: An Optimal Schedule in the TCG

In this chapter, we described the ICG and how to produce the optimal static presentation schedule based on the model using an LP solver. Another important part of scheduling, i.e., run time rescheduling, is developed in the next chapter.

CHAPTER 4 Run Time Rescheduling

A feasible schedule can be derived statically at the compilation time of a multimedia document. A multimedia system then can present the document according to the schedule if the document resides at local sites. However, when the multimedia document is retrieved and presented in a distributed environment consisting of multiple servers and multiple clients, it may not be possible to follow the schedule exactly when the client does not have adequate storage to buffer the complete document before it is presented. If objects are transmitted from media servers to presentation systems at run time, predicting traffic or scheduling can help systems to satisfy the temporal requirements of the document. However, they are not sufficient because of random synchronization errors. Hence a rescheduling is necessary at run time in the distributed system. The events leading to rescheduling are as follows.

Suppose the present time is $t = 0$, and the remaining schedule contains events e_1, e_2, ..., e_n which are to occur at time t_1, t_2, ..., $t_n > t$ respectively. Due to network delay or server delay at $t = 0$, the event e_i will not be able to occur at time t_i as planned (a prediction).

Some assumptions are used to solve run time rescheduling:

- Estimates of network delay, server access time, and client synchronization time are known and reasonably stable over time.
- Server has received the delivery schedule which server must perform and the respective 'deadlines', i.e., they should be done before the 'deadline', or else inform the client system about the problem.

- A server scheduler is responsible for maintaining its deadline schedule such that based on its state, the knowledge of network delay, and the object in the schedule, it could decide if the deadline cannot be met. Once decided, the server scheduler will send a message to the client system informing it about the expected delay.
- Client will invoke a rescheduling upon receipt of a deadline violation.

This chapter explores a dynamic rescheduling approach appropriate for such a distributed system.

4.1 Dynamic Rescheduling in a Distributed Presentation System

4.1.1 Distributed Presentation System Model

The distributed presentation system consists of a set of servers (say S_1, S_2, ...S_k) and a set of clients (say C_1, C_2, ...C_m). Each multimedia presentation involves a single client station and a subset of the servers. We could imagine that a server could be a text server or a video server for example. Hence a presentation involves the retrieval of multimedia objects from the relevant servers and their presentations at the client according to the temporal constraints given in a ICG. It is assumed that when a client starts a presentation session, the multimedia document will be dynamically retrieved from the servers, for example, audio or video clips can be streamed to the client, according to some statically decided presentation schedule.

A presentation session starts with a client invoking the document and hence the associated static schedule. In an ideal environment, the client can send the schedule to the

servers; the servers can access the files and deliver them to the client through the network in time for the presentation. In an actual environment, delays occur at different parts of the system including: (1) server i disk access delay, say ds_i, and (2) network delivery delay from server i to client j, say dn_{ij}. These delays are not necessarily constant and usually depend on the traffic intensity at both the servers and the network. The delay at a server depends on the length of the request queue as well as the actual access delay of the disk. The network delay depends on the contention/bandwidth available and hence the amount of traffic existing on the network.

Even though ds_i and dn_{ij} are variable, for the purpose of scheduling, a client can assume some values of ds_i and dn_{ij} in order to carry out the presentation. Similarly, a server can also assume some value of dn_{ij} in the course of its operation. These two variables can change in time, and when detected, can be made known to the parties concerned.

4.1.2 Retrieval and Delivery Schedules

Thus a presentation involves the cooperation between a client and its servers. The client is responsible for sending its requests to the targeted servers at appropriate times. This set of requests is called a retrieval schedule. The server is responsible for fulfilling the needs of its clients by performing the necessary disk accesses and delivering the data to the respective clients at appropriate times. This set of deliveries for a single client is called the delivery schedule.

Definition: Retrieval Schedule

Given a presentation schedule, the retrieval schedule at a client consists of $\{t_r(obj_i) \mid i = 1,2,\ldots n\}$ where $t_r(obj_i)$ is the latest time to request object i from the server. In the case that object i is a stream object, the request involves getting the first segment of the object from the server, followed by the subsequent streaming at a pre-specified demand rate. Hence, the following relationship holds:

$t_r(obj_i) <= t_{start}(obj_i) - ds - dn$, where ds and dn are the expected server and network delays known to the client.

If equality is used in the above relation, then it is assumed that the request schedule has zero tolerance to the increases of ds and dn.

Definition: Delivery schedule

Given a presentation schedule, the delivery schedule at a server consists of $\{t_d(obj_i) \mid obj_i$ is stored in the server$\}$ where $t_d(obj_i)$ is the time of sending object i to the client. Again in the case of a stream object, this corresponds to the time of sending the first segment of the object onto the network. The following relationship holds:

$t_d(obj_i) <= t_{start}(obj_i) - dn$

Equality can be assumed in the above relation when we apply zero tolerance to the increases of the network delay.

4.1.3 Client-Server Protocol

In starting a presentation session, a client sends its first requests together with the first part of retrieval schedule to the relevant servers. In due course, it expects to receive the data from the servers in time for the presentation according to the static presentation schedule. During the course of the presentation, it will continue sending the rest of retrieval requests to the servers without violating the retrieval schedule.

In return, a server receives all requests from the clients and serves them according to its service policy. Delivery of objects are carried out according to the delivery schedules involved, i.e., attempts are made so that objects are delivered in time. However, failures could occur because of the variable delays at both the server and the network level. When a failure is detected, i.e., a delivery schedule or a presentation schedule violation is detected, the static presentation schedule may no longer be strictly followed. Dynamic rescheduling is invoked.

4.1.4 Dynamic Rescheduling

Dynamic rescheduling is invoked in either of the following situations:

(1) A client cannot perform the event because it has not received the required data.

(2) A server detects a delivery failure because of its local congestion and sends a failure signal to the client.

In both cases, the client becomes aware of the schedule failure and proceeds to compute a new schedule to be used for the rest of the presentation. Suppose TCG' is the remaining part of the TCG yet to be presented. Then the rescheduling problem is to find a remaining schedule for the TCG' given the original presentation schedule. Without loss of generality, we can assume the remaining schedule is $(t_i, t_{i+1}, \ldots t_n)$. Then the rescheduling problem is to find a composite schedule $(t_1, t_2, \ldots t_{i-1}, t_i', t_{i+1}', \ldots t_n')$ that satisfies the TCG, and if no such schedule exists, then derive one that minimizes the number of inconsistencies in the composite schedule.

4.2 Rescheduling Approach

An implicit requirement in dynamic rescheduling is speed. In other words, a new schedule must be computed as quickly as possible. In such a case, QoS optimization may have to be sacrificed for speed. Delays caused by rescheduling may introduce undesirable inconsistencies that cannot be removed by rescheduling. The following example illustrates it.

Consider the example in FIGURE 6. Suppose obj_1, obj_2 and obj_3 are stream objects, and for some reason, the end of obj_1 cannot occur at the scheduled time t = 2. The original presentation schedule is given by (0, 2, 0, 3, 5, 9) corresponding to $(t_{start}(obj_1), t_{end}(obj_1), t_{start}(obj_2), t_{end}(obj_2), t_{start}(obj_3), t_{end}(obj_3))$. Rescheduling is triggered at t = 2. If rescheduling at the client takes too much time, and further delays the end of obj_1 from occurring, say to t = 4, then this will in turn delay the end of obj_2 and may cause inconsistency between the start and the end of obj_2. FIGURE 11 shows the

TCG' after the occurrences of the start of obj_1 and the start of obj_2, and the associated remaining schedule [t_i'].

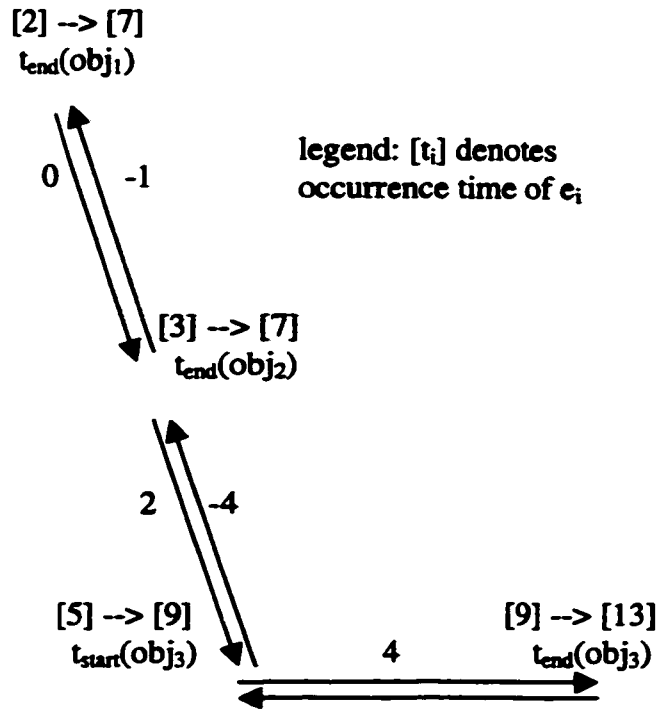


FIGURE 11: New Remaining Schedule

In FIGURE 11, the delay of the end of obj_1 from t = 2 to t = 7 triggers a rescheduling of the end of obj_2, the start of obj_3 and the end of obj_3 leading to a composite schedule (0, 7, 0, 7, 9, 13). Unfortunately, the incurred delay is excessive and the composite schedule (consisting of all events) includes two instances of inconsistencies: the separation time between $t_{start}(obj_1)$ and $t_{end}(obj_1)$ and that between $t_{start}(obj_2)$ and $t_{end}(obj_2)$ become 7, both outside the respective interval [1,2] and [0,6]. If the delivery and rescheduling of the end of obj_1 could be presented at t = 4, then the resulting composite schedule (0, 4, 0, 4, 6, 10) will have only one inconsistency.

There are many possible rescheduling algorithms, leading to various solutions.

4.2.1 Minimal Remaining Time Algorithm (MRTA)

Suppose we ignore potential inconsistencies caused by interval violations between an event that has occurred and an event yet to occur. Then the scheduling of the TCG' using t_i' as the reference start-time can be performed using a simple Minimal Remaining Time Algorithm (MRTA). MRTA involves computing the longest paths in the TCG from e_i to each of the remaining events in the TCG'.

Lemma 4.1: The TCG' is consistent

Proof: The TCG' is a sub-graph of the TCG. Every sub-graph of a consistent graph is consistent. QED

Lemma 4.2: ($t_j' = t_i'$ + longest path from e_i to e_j in the TCG', j = i+1, i+2,...,n) is a feasible schedule for the TCG'.

Proof: Use induction. Suppose the assertion holds for a graph with k nodes. Consider a graph with k + 1 nodes. In computing the time for node (k+1), for each of its predecessor nodes, say node j, we could compute the 'earliest' time to perform (k+1) is at time max $\{(t_j + t_{jk+1}) \mid j$ immediately precedes k+1 in the TCG'$\}$. Taking this value for t_{k+1} will be feasible because otherwise there will be a strictly positive cycle in the TCG'. QED

One difficulty of using MRTA at run time is that the complexity of the above algorithm is quadratic. Hence if the required time for rescheduling is small, such as in

short presentations, then MRTA may be suitable for the presentations with simple constraints. It should be noted that inconsistencies may exist between some e_j ($j > i$) and e_k ($k < i$) because the rescheduling has not considered the constraints between the past and the future events in the presentation.

4.2.2 Linear Shift Algorithm (LSA)

Alternately, the remaining schedule can be obtained by performing a constant shift of each of the remaining times. Specially, suppose $t_i' = t_i + d$; then $t_j' = t_j + d$ for all $j > i$.

Lemma 4.3: $(t_j' = t_j + d \mid j = i+1, i+2, \ldots n)$ is a feasible schedule for the TCG'

Proof: Suppose $t_j' = t_j + d$ and $t_k' = t_k + d$ are the new scheduled times obtained from LSA for any two future events j and k ($j > k > i$). The original scheduled times t_j and t_k satisfy the constraint in the TCG' that $t_j - t_k$ in $(t_{kj}.min, t_{kj}.max)$. Since $t_j' - t_k' = t_j + d - (t_k + d) = t_j - t_k$, $(t_j' - t_k')$ is still in $(t_{kj}.min, t_{kj}.max)$. Hence the new schedule times satisfy the constraint. In other words, it is a feasible schedule for the TCG'. QED

It is simple to implement LSA and efficient to run it. Hence if the required time for rescheduling is small, then LSA may be useful for the presentation with complex constraints. However, like MRTA, LSA totally ignores temporal relationships between past and future events, and will very likely generate many inconsistencies because of the constant delay d applied to all future events.

4.2.3 Minimal Relaxation Algorithm (MRA)

Interval constraints in an ICG are like rubber bands, and a schedule in them is just one state of rubber bands. Other states of these rubber bands can also be feasible schedules for the ICG. The idea of the minimal relaxation algorithm (MRA) is to stretch the remaining interval rubber bands in the TCG' minimally so that they do not break and hence are consistent, given t_i' caused by the delivery delay associated with e_i. We use pred(j) = {i | (i,j) is in the ICG}, i.e., the immediate predecessor events of j in the ICG.

for j = i +1 to n do

$$t_j' := \max_{k \text{ in pred } (j)} \{ t_j, t_k' + t_{kj}.\min \};$$

end;

The MRA allows future events to be minimally postponed from its original schedule t_j. By doing so, it is likely that fewer inconsistencies may occur, as fewer changes are made to the original schedule. The complexity of this algorithm is linear to the size of the TCG' and hence is cheaper than MRTA.

Inconsistency occurs when the refinement of t_j' leads to a situation where for some k in pred(j) such that $t_j' - t_k'$ (or t_k) > $t_{kj}.\max$. In other words, through another causal path, t_j' has to be postponed beyond the path from e_k to e_j.

In optimizing the number of inconsistencies, the MRA can be slightly modified so that instead of simply choosing the maximum path to e_j, we choose among all paths to e_j the one which minimizes the number of inconsistencies so created.

MRA:

for j = i +1 to n do

$\{c_{jk}\} := \{ \max \{ t_j, t_k' + t_{kj}.\min \} \mid k \text{ in pred}(j)\}$;

$t_j' := \max_{k \text{ in pred (j)}} \{ c_{jk}\}$ with the minimal number of inconsistencies;

end;

4.2.4 Guaranteed Rescheduling

In the previous sections, we have discussed some rescheduling algorithms such as MRTA, LSA and MRA. But all of them can not guarantee a fully consistent rescheduling at run time. In this section, we try to find a suitable way for guaranteed rescheduling, that is, finding a schedule without any inconsistencies.

Given:

(1) A TCG and a remaining schedule ($t_1 = 0, t_2,\ldots, t_k$) for events (nodes) E' = $\{e_1,e_2,\ldots,e_k\}$ in the TCG and

(2) Residual temporal constraints in E', such that for e_j in E': $0 <= t_j <= t_j.\max$

Note that $t_j.\max >= 0$ because e_1 is the first event in the list whose data delivery is delayed. For the schedule to be feasible up to this point, all constraints of the original TCG must be satisfied, and hence $t_j > 0$.

The problem here is whether we can derive a feasible schedule taking care of the delayed event e_1. If we can, how to derive the feasible schedule.

Solution:

Suppose in deriving the feasible schedule (t_1, t_2,..., t_k), we have also computed the longest paths between all nodes, i.e., ($\bullet_{i\,j}$) represents the longest path from node i to node j in the TCG. From previous discussion, we know $\bullet_{ij} + \bullet_{ji} <= 0$. Otherwise a positive cycle would result and the TCG is inconsistent.

We could take care of the residual temporal constraints by incorporating them into the TCG to form a new TCG' as follows:

For each $t_j <= t_j.max$, we augment the TCG with the edge (e_j, e_1) that is labeled with $-t_j.max$.

Property: The TCG' may become inconsistent.

Reason: Inconsistency could occur when $\bullet_{1\,j} - t_j.max > 0$.

The consistency of the TCG' can be checked by asking: for each edge (e_j, e_1) added to the TCG', is the condition $\bullet_{1\,j} - t_j.max <= 0$ satisfied ?

Lemma 4.4: If $\bullet_{1\,j} - t_j.max <= 0$ for every e_j in E', then the TCG' is consistent.

Proof: Suppose a positive cycle in the TCG' involves more than one of the edges used to augment the TCG. This is a contradiction because all these edges end at e_1 (hence not a cycle). Hence we need only test each of these new edges separately using the above condition. QED

Suppose the TCG' fails the condition in a subset of the new edges introduced. Accordingly, there is not much that could be done to avoid such inconsistencies. The only

reasonable alternative is to remove these violation edges from the TCG' to get the corresponding TCG" and then try to derive a feasible schedule for the TCG". The challenge here is to derive such a schedule as quickly as possible, reusing information from $\{ \bullet_{ij} \}$ and $\{ t_j.max \mid j \text{ in } E" \}$.

We know from the Lemma 4.2 in SECTION 4.2.1 (Minimal Remaining Time Algorithm) that for a consistent TCG, a feasible (shortest presentation) schedule is obtained by computing (the length of) the longest path from e_1 to every node.

Lemma 4.5: $(t_i" = \bullet_{1i}" \mid i = 1,2, \ldots k)$ is a feasible schedule of the TCG".

Proof: From the definition of the TCG", there are no violation edges in the TCG", so it is a feasible schedule of the TCG". QED

Hence the challenge is to compute $\bullet_{1i}"$ as fast as possible, reusing data already available. The following result is perhaps surprising:

Lemma 4.6: $\bullet_{1i}" = \bullet_{1i}$

Proof: Suppose $\bullet_{1i}" > \bullet_{1i}$. So the longest path from e_1 to e_i goes through one of the new edges in the TCG". But this is not possible as each of those edges ends at e_1. QED

The above surprise result indicates that the MRTA is optimal for quick presentations and every event would occur at the earliest time allowed given the temporal constraints because the schedule computed by MRTA is from the longest paths. When an event, say t_i, is delayed by d time units, the above lemma asserts that the new remaining

schedule can be simply computed as (t_i + d, ..., t_n + d), i.e., apply linear shift to the schedule obtained from MRTA. Or equivalently, it is optimal to use LSA on MRTA.

MRTA does not take QoS into consideration. Hence it may not give a good presentation. On the other hand, the optimal (QoS) algorithm leads to a schedule which may not minimize inconsistencies when rescheduling is applied.

Since we have decided to adopt QoS_OPT which represents the strategy discussed in SECTION 3.2 (Production of an Optimal Static Schedule) to produce an optimal quality schedule statically, then the next challenge is how to reschedule presentations based on QoS_OPT.

4.2.4.1 Retime Events on MRA (Minimal Relaxation Algorithm)

Assumption: A feasible TCG" consisting of nodes $\{e_1,e_2,\ldots,e_k\}$ and an original schedule $(t_1,t_2,\ldots,t_k)$ which cannot be maintained as e_1 must be delayed to $t_1' > t_1$. From earlier work, we know that TCG" is a multi-graph, i.e., there could be multiple edges between two nodes.

Define $t_i^* = t_i + (t_1' - t_1)$. Hence t_i^* is the translated schedule of event e_i relative to the occurrence of e_1 at t_1'. Notice that $t_1^* = 0$.

The idea of MRA is to retime e_i from t_i^* to $t_i^\wedge$ only if inconsistency occurs in one of its edges relative to another event, say e_j. There are two possible consequences:

(1) $e_i \rightarrow e_j$:

If $t_j^* - t_i^* < t_{ij}.min$ then $t_i^\wedge := t_j^* - t_{ij}.min$

If $t_j^* - t_i^* > t_{ij}.max$ then $t_i^\wedge := t_j^* - t_{ij}.max$

As a consequence of the above, $t_j^* - t_i^\wedge$ is at $t_{ij}.min$ or $t_{ij}.max$.

(3) $e_j \rightarrow e_i$:

If $t_i^* - t_j^* < t_{ji}.min$ then $t_i^\wedge := t_j^* + t_{ji}.min$

If $t_i^* - t_j^* > t_{ji}.max$ then $t_i^\wedge := t_j^* + t_{ji}.max$

As a consequence of the above, $t_i^\wedge - t_j^*$ is at $t_{ji}.min$ or $t_{ji}.max$.

Now let us turn our attention to the effect of delay t_1 to t_1'. If all edges connected to e_1 remain consistent, i.e., t_{i1} or t_{1i} lies in $[t_{i1}.min, t_{i1}.max]$ or $[t_{1i}.min, t_{1i}.max]$, then the schedule remains feasible and no retiming is invoked.

Without loss of generality, say $e_1 \rightarrow e_2$ and $t_2^* < t_{12}.min$ (this is the only possible inconsistency caused by the delay of e_1), then e_2 has to be delayed as well. Minimal relaxation will assign t_2^* to $t_{12}.min$. Hence t_2^* is at the minimum of a forward edge.

Suppose through this retiming of t_2^*, an inconsistency occurs between $e_2 \leftarrow e_3$, i.e., $t_2^* - t_3^* > t_{32}.max$ (again this is the only possible inconsistency, as t_2^* is increased, and not decreased in the retiming), then the retiming of e_3 leads to t_3^* to $t_2^* + t_{32}.max$. Hence t_3 is at the maximum of a reverse edge from e_2 to e_3.

From the above, if we could trace a cycle of retimed events, then it must be that we could trace a cycle involving forward edges in the minimum and reverse edges at the maximum, or equivalently, an inconsistent cycle (strictly positive cycle) in TCG". The contradiction implies that a node cannot be retimed more than once. Hence termination.

We give an example to explain the idea as follows.

Suppose an ICG shown in FIGURE 12 consisting of nodes $\{e_0, e_1, e_2, e_3, e_4\}$ and an original schedule (0, 1, 4, 6, 10) corresponding to $(t_0, t_1, t_2, t_3, t_4)$, where t_i is the time of occurrence of e_i. Suppose due to network delay or server delay, e_1 has to be delayed by 2, then the schedule becomes (0, 3, 4, 6, 10) shown in FIGURE 13.

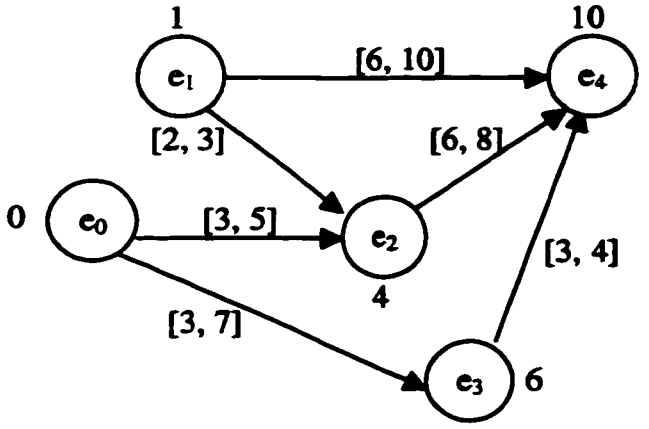


FIGURE 12: The ICG to Show Retiming Idea

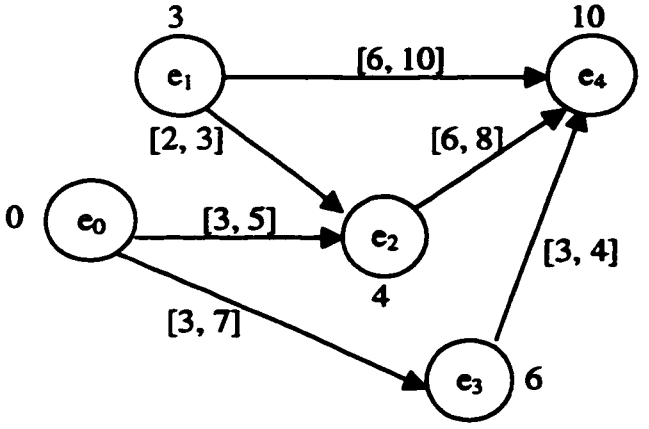


FIGURE 13: Inconsistent Schedule due to Delay Corresponding to FIGURE 12

Notice that the duration from $e_1 \rightarrow e_2$ is 1 in FIGURE 13; it violates the corresponding interval constraint [2, 3]. At this moment e_0 has happened, then we need to reschedule the remaining schedule (t_1, t_2, t_3, t_4).

Before applying MRA to retime events, we need to transform the constraints between the past events (for example e_0) and the future events (for example e_2 and e_3) into a new graph shown in FIGURE 14. We merge the two edges from $e_1 \rightarrow e_2$ in FIGURE 14 and get a graph shown in FIGURE 15.

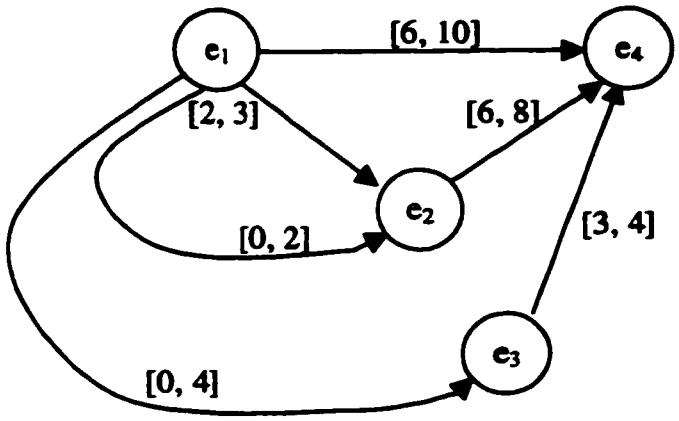


FIGURE 14: Transformed ICG from ICG in FIGURE 13

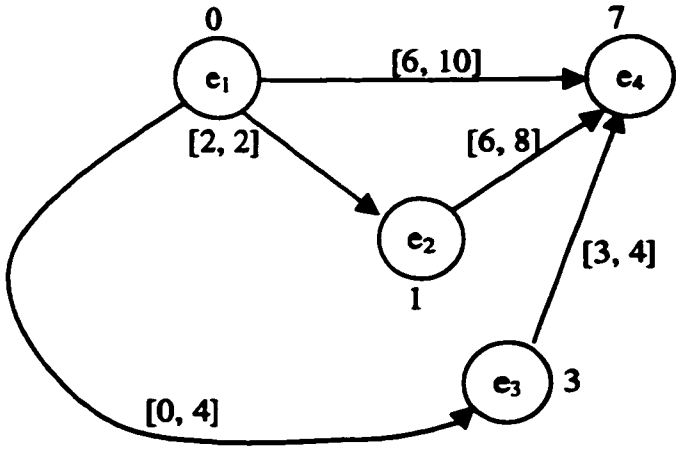


FIGURE 15: Merged Edge ICG from FIGURE 14

At this moment, suppose $t_1 = 0$, then remaining schedule is (0, 1, 3, 7) shown in FIGURE 14 corresponding to (t_1, t_2, t_3, t_4), and there is inconsistency between e_1 and e_2.

We retime e_2 from 1 to 2 (Minimum of the edge from e_1 to e_2) shown in FIGURE 16, and new inconsistency occurs from e_2 to e_4. Similarly, we retime e_4 from 7 to 8 (Maximum of the edge from e_2 to e_4) shown in FIGURE 17, and new inconsistency occurs from e_3 to e_4. Finally we retime e_3 from 3 to 4 shown in FIGURE 18 and reach a cycle, hence terminate rescheduling. The final consistent schedule (0, 2, 4, 8) is shown in FIGURE 18.

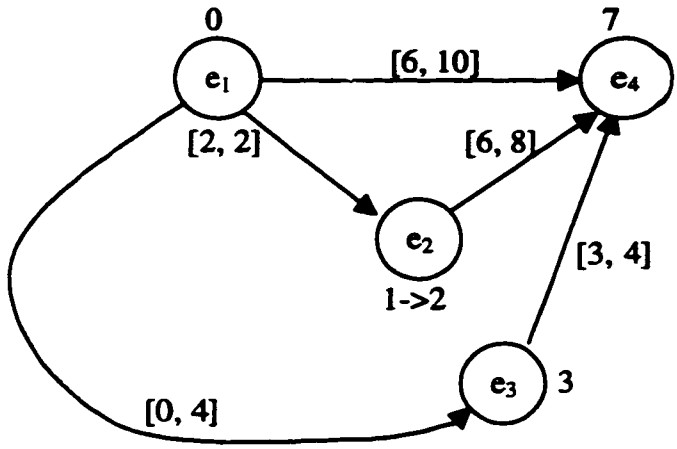


FIGURE 16: Retiming e_2 from 1 to 2

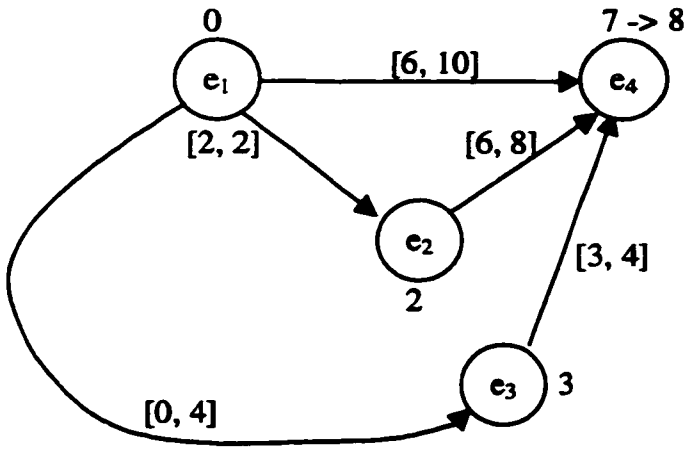


FIGURE 17: Retiming e_4 from 7 to 8

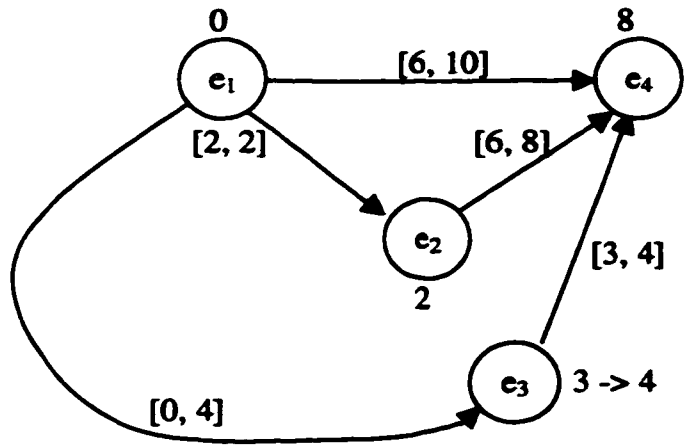


FIGURE 18: Retiming e_3 from 3 to 4 and Final Consistent Schedule

4.3 Rescheduling Based on Retiming Method

In SECTION 4.2 on Rescheduling Approach, we have discussed and proved the rescheduling idea, i.e., applying MRA to retime events. This section describes the rescheduling algorithm using this idea.

Definition: A Feasible Schedule in an ICG

Given an ICG G = (N, A, T), a feasible schedule of G is given by G' = (N, A, T'), where T' = $\{t_{ij}^*\}$ such that $t_{ij}.min <= t_{ij}^* <= t_{ij}.max$. Also, t_{ij}^* is called the weight of the edge (e_i, e_j) where e_i, e_j in N.

Definition: The weight of a directed cycle

Given a feasible schedule G' = (N, A, T'), where N, A, T' are the set of nodes, edges and the weights of edges respectively. The weight of a directed cycle C = $e_1, e_2, \ldots, e_k, e_{k+1}$ ($e_i \in N, e_{k+1} = e_1$) is defined as :

$w(C) = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$, where x_i is the weight of (e_i, e_{i+1}) and

$$a_i = \begin{cases} 1 & \text{if } (e_i, e_{i+1}) \text{ has the same direction as C} \\ -1 & \text{otherwise} \end{cases}$$

Definition: Balanced graph, Cycle equations

Given a feasible schedule G' = (N, A, T'), G' is said to be balanced if for any cycle C, w (C) = 0. If G' is balanced, w (C) = 0 is said to be a cycle equation of C.

4.3.1 Retiming Method

The retiming method is the basis of the Rescheduling Algorithm and can guarantee the correct use of the idea discussed in SECTION 4.2 (Rescheduling Approach). Also it can keep the balance of a feasible schedule graph after changing the weights of some edges. In other words, applying retiming method properly may keep the consistency of the ICG. That may produce another feasible schedule further by the Rescheduling Algorithm.

Given an ICG G = (N, A, T), a feasible schedule of G is given by G' = (N,A,T'), where T' = $\{t_{ij}^*\}$ such that $t_{ij}.min <= t_{ij}^* <= t_{ij}.max$. Note that G' is a balanced graph.

For any node $e \in N$, suppose $x_1, x_2, \ldots x_m$ are the weights in the feasible schedule G' associated with the edges $(e_1, e), (e_2, e), \ldots, (e_m, e)$ respectively, such that $(e_1, e), (e_2, e), \ldots, (e_m, e)$ are all incoming edges to e, where $x_i \in T'$, $(e_i, e) \in A$ and m is a positive integer.

Suppose $y_1, y_2, \ldots y_m$ are the weights in the feasible schedule G' associated with the edges $(e, e_1), (e, e_2), \ldots, (e, e_n)$ respectively, such that $(e, e_1), (e, e_2), \ldots, (e, e_n)$ are all outgoing edges from e, where $y_i \in T'$, $(e, e_j) \in A$ and n is a positive integer.

Retiming Algorithm:

With a given offset value d, a new feasible schedule of G, say G" = (E, A, T"), can be obtained by changing the feasible schedule G' in such a way that $x_1, x_2, \ldots x_m$ is replaced by $(x_1+d), (x_2+d), \ldots, (x_m+d)$ respectively, and $y_1, y_2, \ldots y_n$ is replaced by $(y_1-d), (y_2-d), \ldots, (y_n-d)$ respectively such that $t_{ij}.min <= (x_i+d), (y_j-d) <= t_{ij}.max$, where $(x_i+d), (y_j-d) \in T"$, and d can be a real number.

Lemma 4.7: G" = (E, A, T") is also a balanced feasible schedule.

Proof: Every cycle that passes through the common node e must have to traverse two of these edges. One of the two edges must be the incoming edge, and other must be the outgoing edge. Therefore, every case of the traversal will not break the balance of the cycle. QED

4.3.2 Rescheduling Algorithm

Suppose the present time is t = 0, and the remaining schedule contains the events $e_0, e_1, e_2, \ldots e_k$. The event e_0 should happen at t = 0 at the client system, but it has to be delayed with the value delay (e_0) = d after receiving a message from the server or

network system. Hence, some of the edges connected to e_0 are relabeled with a change of weight by d. This change may cause inconsistency in the sense that the retiming increases or decreases some of the weights to beyond what is allowed in their temporal constraints. Here the idea of retiming is done locally at a single node only. If the weights of some edges to or from e_0 cannot be adjusted with d, it is still possible to propagate the residual amount to the associated events e_i. For example, suppose the weight between e_0 and e_1 can be changed by $d' < d$. The remaining weight (d-d') can be assumed to be removed from e_0 to e_1, and hence retiming at e_1 with the weight (d-d') should take place during the next iteration. We restrict the propagation so that it does not introduce more inconsistency than originally caused by the delay. In short, the retiming can be done by propagation from an event node to another, and the propagation will be continued as long as the number of inconsistencies does not grow.

Rescheduling Algorithm:

Input:

- An ICG G = (N, A, T)
- Current feasible presentation schedule G' = (N, A, T')
- Delay of event e_0: delay(e_0)

Output:

- A new feasible schedule G" = (N, A, T") with minimizing inconsistencies after e_0 is delayed by delay(e_0)
- A set of values of inconsistencies (Default value zero means it's consistent)

1. Let $e_k = e_0$, d = delay(e_0).

2. Retime e_k with d as offset on G' getting G" using Retiming Algorithm in SECTION 4.3.1.

3. Let G' = G".

4. If G" does not cause new inconsistencies, then report no inconsistency and halt.

5. While there exists an inconsistent edge (e_k, e_j) in G' do

i. Let $in(e_j) = \{ t_{ij} \mid (e_i, e_j) \in A, i \neq j$ and incoming to e_j, $t_{ij} \in T'$ is the feasible schedule of $(e_i, e_j) \}$.

Let $out(e_j) = \{ t_{ji} \mid (e_j, e_i) \in A, i \neq j$ and outgoing from e_j, $t_{ji} \in T'$ is the feasible schedule of $(e_j, e_i) \}$.

ii. Let $d_1' = \min_{t_{ij} \in in(e_j)} \{ (t_{ij} - t_{ij}.min) \mid t_{ij} \in T', t_{ij}.min \in T \}$.

Let $d_2' = \min_{t_{ji} \in out(e_j)} \{ (t_{ji}.max - t_{ji}) \mid t_{ji} \in T', t_{ji}.max \in T \}$.

Let $d' = \min \{d_1', d_2'\}$.

iii. If $d' >= d$, then /* eliminate the delay */

a. Retime e_j with d as offset on G' to get G" using Retiming Algorithm in SECTION 4.3.1.

b. Let G' = G" and go to step 5.

iv. Otherwise($d'<d$), if e_i has occurred, then /* minimize the delay */

a. Retime e_j with d' as offset on G' to get G" using Retiming Algorithm in SECTION 4.3.1.

b. Record inconsistent value as $d - d'$.

c. Let G' = G" and go to step 5.

v. Otherwise, /* propagate the delay */

a. Retime e_j with d as offset on G' and get G" using Retiming Algorithm in SECTION 4.3.1. /* new inconsistencies occur.*/

b. Let $d = d - d'$ and let $e_k = e_j$.

c. Let G' = G" and repeat step 5.

6. Report information about consistencies and halt.

This algorithm is greedy to minimize the number of inconsistencies caused by the delay. It tries not only to keep the consistency of the remaining part of an ICG, but also to satisfy the constraints between the past and future events.

4.3.3 Examples

Example 1:

Consider a multimedia document consisting of three stream-objects obj_1, obj_2 and obj_3 that are modeled by a temporal constraint graph in FIGURE 9. The optimal presentation schedule shown in FIGURE 10 can be produced from an LP solver, and the corresponding timeline model of the presentation schedule is in FIGURE 19. If a client detects a synchronization error at t = 3 and finds the end of obj_2 has to be rescheduled with $delay(t_{end}(obj_2)) = 1$, the rescheduling algorithm can produce a new presentation schedule as in FIGURE 20. The corresponding timeline model is in FIGURE 21.

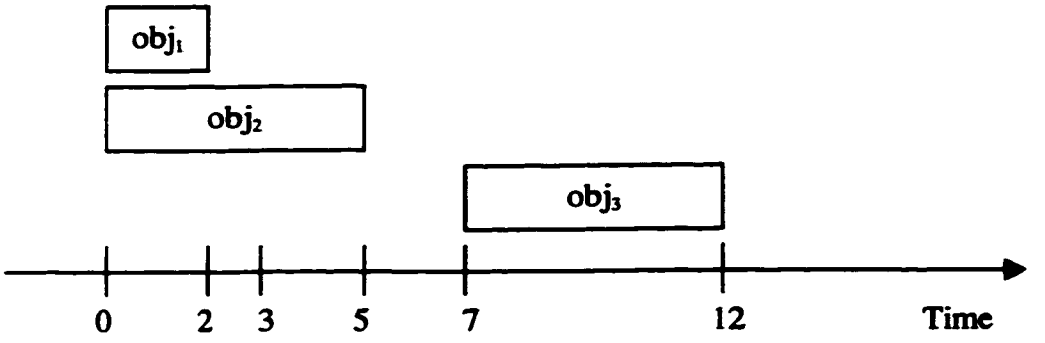


FIGURE 19: The Timeline of the Optimal Presentation Schedule of the Document

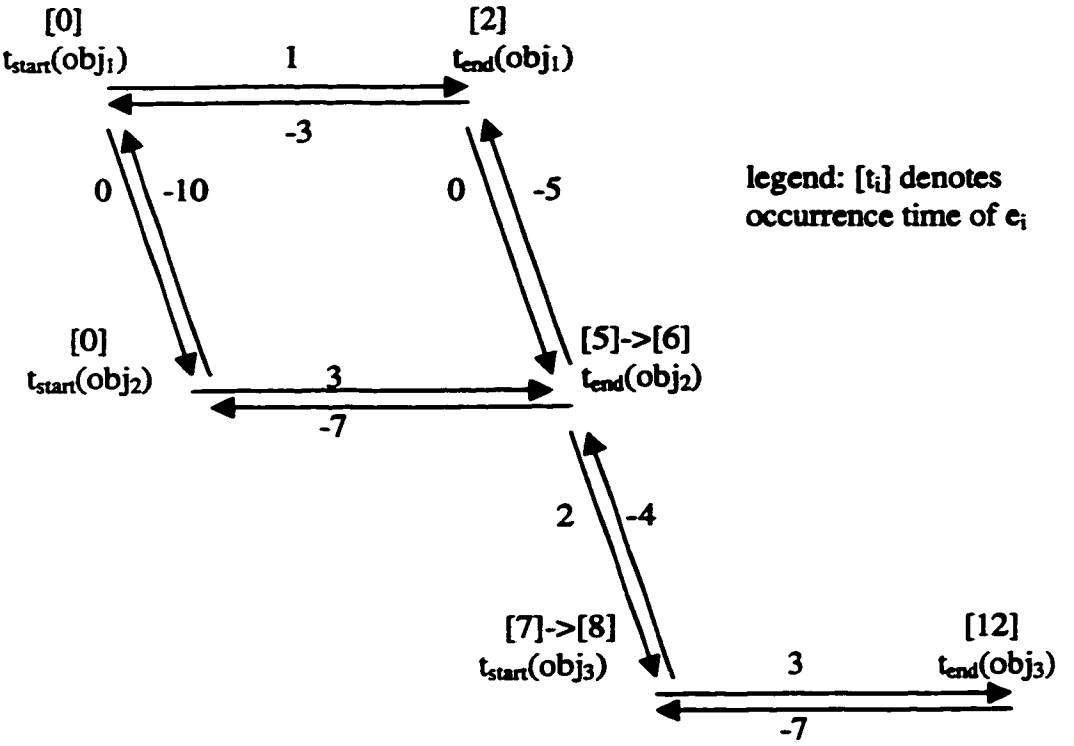


FIGURE 20: A Rescheduled Presentation Schedule (Eliminates delay($t_{end}(obj_2)$) = 1)

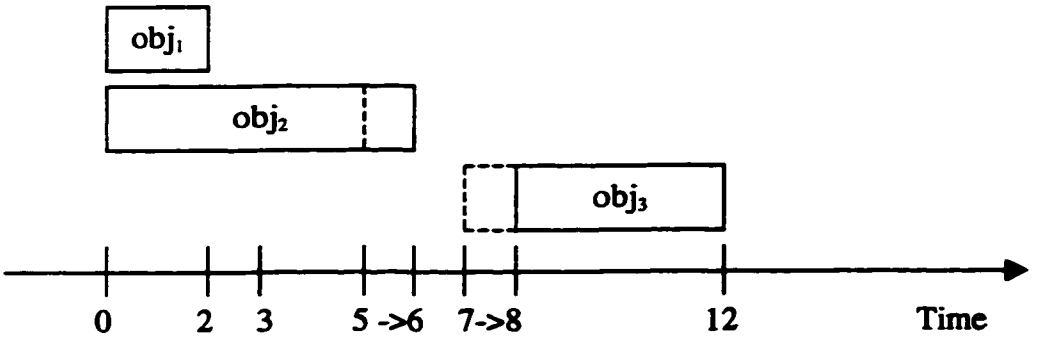


FIGURE 21: Timeline of the Rescheduled Presentation Schedule (Eliminates Delay)

Example 2:

From example 1, if we change delay($t_{end}(obj_2)$) from 1 to 3, then the rescheduling algorithm can not derive a new presentation schedule that eliminates inconsistencies. But it can produce a presentation schedule that minimizes the inconsistency to 1 as in FIGURE 22 and FIGURE 23. If the end of obj_2 occurs at t = 7, the presentation is consistency. In this example, obj_2 has to occur at t = 8; then the inconsistent number is 1.

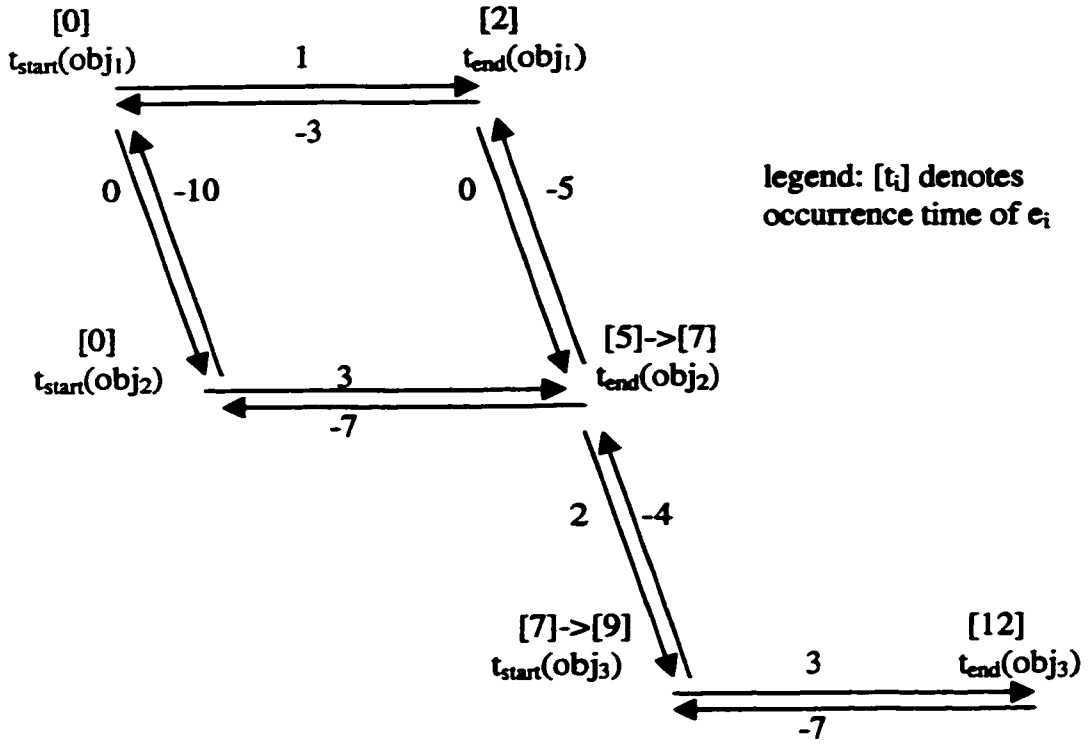


FIGURE 22: A Rescheduled Presentation Schedule (Minimizes Inconsistency to 1)

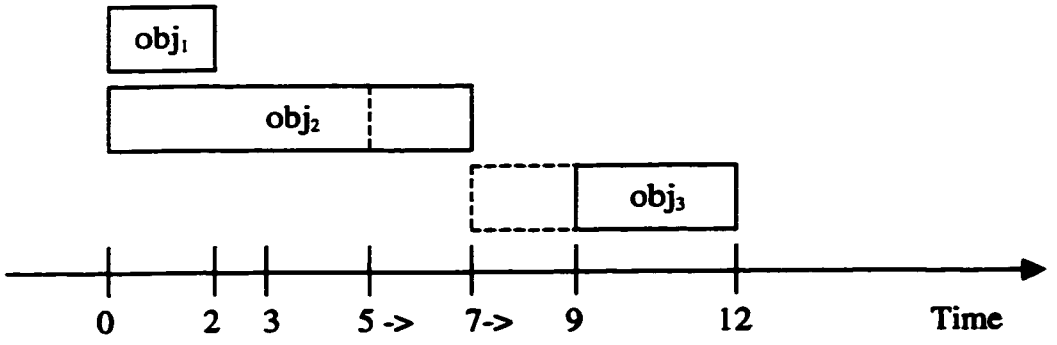


FIGURE 23: The Timeline of the Rescheduled Schedule (Minimizes Inconsistency to 1)

In this chapter, we described an effective algorithm for guaranteed rescheduling at run time. With this we have completed the development of each part of a synchronization scheduler. In the next chapter, we introduce how to put such a scheduler into a distributed environment.

CHAPTER 5 Design of a Scheduler for a Distributed Multimedia Presentation System

In CHAPTER 3 and CHAPTER 4, we developed some techniques to produce the optimal presentation schedule and to reschedule the presentation at run time. These techniques are the most important parts for a synchronization scheduler. If we consider a concrete multimedia presentation system, in particular in a distributed environment, there still are many factors that should be taken into consideration. In this chapter, we first introduce some basic ideas about the design of a synchronization scheduler for a distributed multimedia presentation system. Then we outline the architecture and show the relationship between different parts of the system components. Finally, we describe several important parts of the system in brief.

5.1 The Scheduler

In a distributed environment, multimedia documents can reside in many media servers. Media objects can be of different types such as text and video, and can be either atomic or stream in nature. To present such a multimedia document, the media objects composing the document have to be retrieved properly from their storage sites acting as servers over a computer network to the retrieving system acting as a client. If the retrieval process is initiated by the client system, the object retrieval is composed of the following phases:

- Identify a presentation schedule that satisfies the temporal specification associated with the multimedia document.

- Identify a delivery schedule that specifies the time instant at which the server system should deliver the media objects to the client system.
- Dynamically change the presentation schedule if necessary at run time.

Actually, the function to complete the above three phases constitutes the scheduling of the presentation in the multimedia presentation system, i.e. the scheduler. Therefore, the key part in the architecture of a multimedia document presentation system is the scheduler, which produces the time instants and durations of the presentation and the delivering time instants of multimedia objects at both online time (run time) and offline time (before run time).

There are two types of schemes [LLG94] for the scheduling of media objects in a distributed multimedia system: the centralized scheduling and the distributed scheduling. In the centralized scheduling strategy, we assume that the scheduler manages the whole temporal knowledge including temporal constraints in the client system. On the other hand, in the distributed scheduling strategy, each media server co-operates in the scheduling process by exchanging its own temporal knowledge. Due to the fact that the implementation of the distributed scheduling strategy is much more complicated, we prefer to adopt the centralized scheduling strategy in our architecture.

The scheduler consists of three parts: the presentation scheduling unit, the delivery scheduling unit and the run-time rescheduling unit. Each part carries out the corresponding functions of the above three phases respectively.

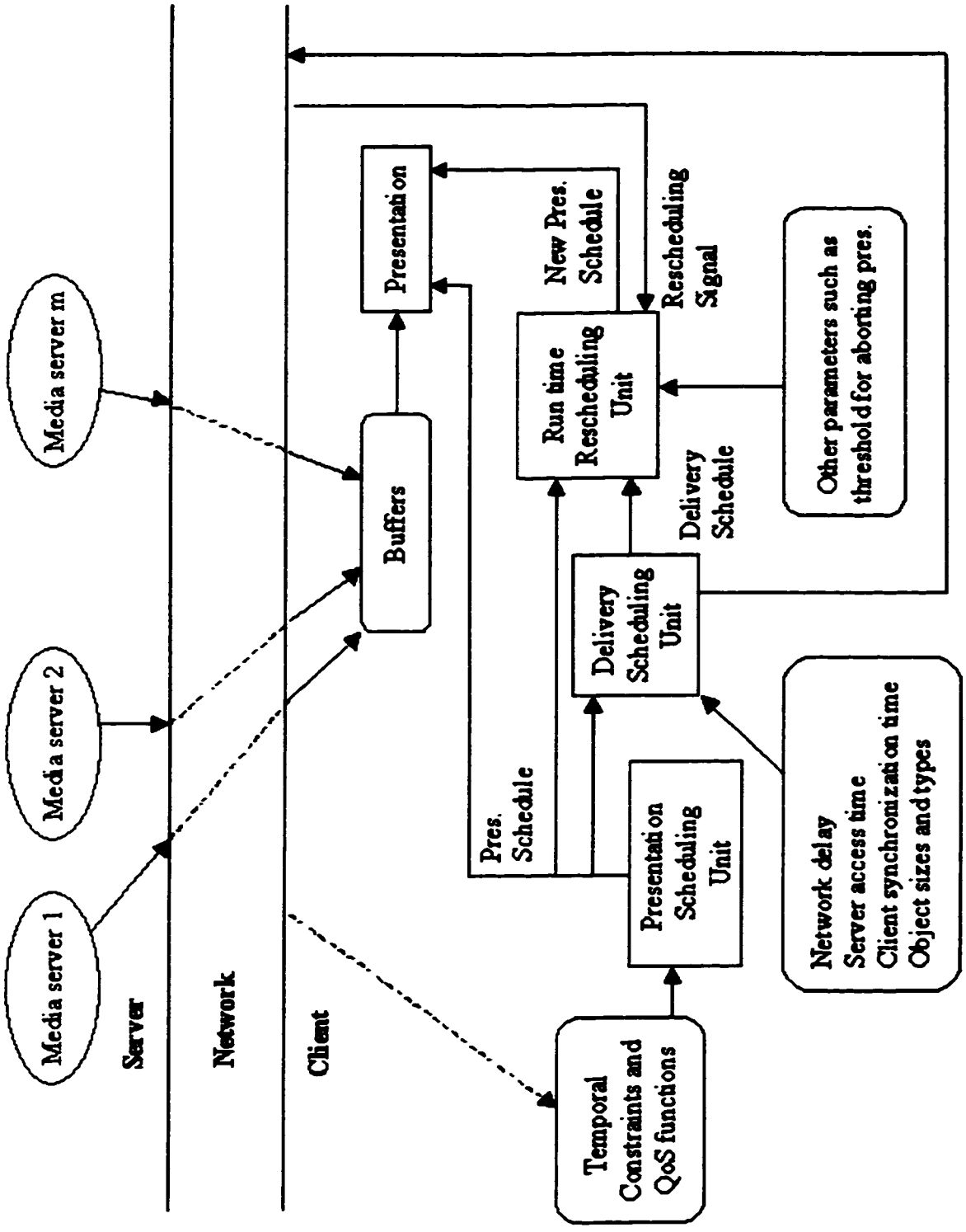


FIGURE 24: Scheduler in a Centralized Architecture of Multimedia Presentation System

5.2 The Architecture of a Distributed Multimedia Presentation System

FIGURE 24 shows the scheduler in the centralized architecture of a multimedia presentation system. First, the temporal constraints associated with a given multimedia document and a group of QoS functions are used by the presentation scheduling unit to generate an optimal presentation schedule. In addition, based on this presentation schedule and the estimated values of network delays, server access times and client synchronization times, the delivery-scheduling unit determines a delivery schedule. Furthermore, at run time, the servers detect synchronization errors. In other words, the servers are responsible for predicting whether or not a scheduled event can occur on time in the client system. If not, the servers will inform the run time rescheduling unit in the client system to adjust the presentation schedule such that it can be satisfied with the temporal relationship of the multimedia document. Also, the client system may need to detect synchronization errors because of the network delays and inform the run time rescheduling unit to adjust the presentation schedule. Finally, the run time rescheduling unit will recompute the appropriate presentation schedule to eliminate or minimize the inconsistencies caused by server delays and network delays.

It is important to note that, although the client system is responsible for the scheduling in the centralized architecture, it still needs the cooperation from the servers, which indicate rescheduling events.

The input and output of the centralized multimedia presentation system are as follows:

Input: The input to the system consists of:

- A set of temporal constraints.

- A set of QoS functions.
- The estimated values of network delays, server delays and client synchronization times.
- A set of object sizes and object types.
- Other parameters such as the threshold (percentage of the deadline) for aborting the presentation for inconsistent media objects.

Output: The output is presentation schedules and delivery schedules, which satisfy input requirements.

5.3 Multimedia Presentation System Components

5.3.1 Presentation Scheduling Unit

The presentation scheduling unit computes an optimal presentation schedule for the given temporal constraints and the corresponding QoS functions using a linear programming (LP) solver; it takes maximizing the cumulative QoS value as the objective. Because the temporal specification cannot be used as the input of the LP solver directly, it has to be translated into appropriate forms such that they can be used as the input of the LP solver. The detailed process is developed in CHAPTER 3, Optimal Static Presentation Schedule.

5.3.2 Delivery Scheduling Unit

On the basis of presentation schedules, object sizes and types, estimated values of network delays, server access times, and client synchronization times, the delivery-scheduling unit computes a delivery schedule.

The computation depends on object types because of two reasons. First, different types of media objects come from different media servers with different characteristics, such as server access times. Second, atomic objects and stream objects have different temporal requirements in order to be presented adequately. In fact, atomic objects must arrive completely at the buffer of the client before the presentation, while in most cases stream objects require some fraction to arrive at the client buffer before the presentation.

Although constructing a delivery schedule is an essential function of a distributed multimedia presentation system, it is not the emphasis of our work due to the fact that it is dependent on system features rather than how to schedule the multimedia presentation. Some relevant discussions and concrete techniques are described in SECTION 2.4.2, Object Delivery and Presentation in a Multimedia Presentation, and in SECTION 6.3.7, Delivery Scheduling Unit.

5.3.3 Run Time Rescheduling Unit

In order to satisfy the temporal constraints of a multimedia document at run time, the client system needs to reschedule the presentation. The flexibility in the temporal constraints of the multimedia document may help in deriving a feasible presentation schedule that meets temporal requirements. In fact, the run time rescheduling unit tries to

modify the presentation schedule using a retiming method in the centralized multimedia presentation system.

This module takes as input delay requirements and the current presentation schedule graph that is an interval constraint graph with the edges labeled by the presentation schedule instead of the original edge labels. It outputs a new presentation schedule with the corresponding presentation schedule graph as the input of the next rescheduling or it reports consistencies if the retiming method fails to eliminate them. In addition, the run time rescheduling unit sends the presentation schedule to the presentation system.

In summary, at run time, the rescheduling unit recomputes presentation schedules such that they can be satisfied with the temporal constraints of a multimedia document. More description can be seen in CHAPTER 4, Run Time Rescheduling.

In this chapter, we introduced the general design of a synchronization scheduler in a distributed environment. In order to validate the usefulness and effectiveness, we implemented our scheduler in a simulated distributed environment. In the next chapter, we describe some important details of our implementation. Many strategies adopted in our implementation can be different in other implementations, but the basic ideas are as shown in this chapter.

CHAPTER 6 Implementation in a Simulated Distributed Environment

Our contributions in this thesis are (1) a significant QoS Model and corresponding optimal presentation schedule and (2) an effective and more adaptive rescheduling strategy based on temporal constraint graphs. To validate the usefulness and effectiveness in a distributed scheduler for multimedia presentations involving multiple servers and multiple clients, we implemented our scheduler in a simulated distributed environment. Then, we can design, run and analyze some simulation experiments.

To simplify the problem, we simulate the running environment that includes the client simulator, the network simulator, and the server simulator. They simulate the clients, the network, and the servers respectively for multimedia presentations in a distributed system. The real running environment may be more complicated than that we simulated, but our simulators mainly reflect the dynamic features of the system. Therefore, although some simulated parameters and processes seem over simplified, they do make sense for our purpose. In short, the implementation focuses on implementing the functions of the scheduler.

The implementation is on solaris 2.5, SUNW, Ultra-2(sparc) using C language, and the linear programming solver used is implemented by Jeroen Dirks.

This chapter also describes some important details of the implementation in the simulated environment.

6.1 Simulation of a Distributed Environment

6.1.1 Dynamic Information

To simulate the dynamic execution of the scheduling system, we need to keep individual records or status for each server, network, and client. The status consists of all information for the simulation and changes dynamically. For example:

- Server status: includes delivery schedules, server traffic.
- Network status: includes network traffic, the number of objects on network.
- Client status: includes presentation schedule, buffer status.

Also, the simulation is based on events, and the occurrences of events cause changes in the system. To capture those changes, individual components in the system need to dynamically exchange the information relevant to the events. In this way, the scheduler can work in a simulated environment that has a similar dynamic behavior of a real distributed environment.

6.1.2 Global Clock

A key issue in the simulation is how to deal with time since the synchronization refers to time. In a real distributed system, a global clock is often provided to coordinate the times between senders and receivers because conventional system clocks are unstable, hence unsuitable.

The real global clock system is very complicated. Fortunately, our simulation is based on a local system, so we just set a virtual global clock to accord all clocks in the simulated system. This virtual global clock has an initial value of zero and in one turn increases one unit that represents one second. In each turn, servers, network, and clients check and change their dynamic status respectively.

6.1.3 Server Simulation Model

In our server simulation model, servers can offer services to multiple clients at the same time. We give some assumptions as follows.

(1) Servers can deliver multiple media objects at the same time, and the order of delivering these objects is given by the server scheduler which is responsible for managing a request queue. Hence the traffic generated by an object being delivered is modeled by the size of the object. Stream objects are modeled by a sequence of objects with a pseudo-periodic arrival rate.

(2) While the data units of an object are delivered, the object being delivered is active until the data units from the present object are completely delivered from the secondary storage. After that, the object becomes inactive. Then based on the delivery schedule, the next object can be handled.

Model:

- Server throughput available per active object is given by server throughput / the number of active objects in delivery

- Server delay time is given by size of object / server throughput available per active object.

Two events affect the server status. One is an object becoming active. The other is that an active object completely finishing its delivery at the server and becoming inactive. When one of these two events happens, the server throughput available per active object changes dynamically because the number of active objects in delivery is a variable parameter in time. Hence our simulation model is a time and event driven dynamic model. Whichever event happens, the server needs to recompute the server throughput available per active object and the server access time because the event may cause two changes. One is the number of active objects in delivery, and the other is the size of remaining parts of delivered objects.

In the recomputation, first the size of object needs to be replaced by the size of remaining parts of the object: size of object – dt*server throughput available per active object, where dt is the duration from the time last event happens until the time current event happens. Second, the server throughput available per active object is recomputed with a new value of the number of active objects in delivery. Finally, the new server access time needs to be recomputed for each delivered object.

6.1.4 Network Simulation Model

In our network simulation model, the network connects a set of sites that are either servers or clients. We assume that the features of clients do not affect the network traffic, so servers dictate the network traffic. Furthermore, we also assume the following.

(1) Network can serve multiple servers and transmit multiple media objects from one server at the same time. Objects from servers have similar characteristics, and all objects are given equal priority of access to the network. Hence the traffic generated by a server sending an object is modeled by the size of the object. Stream objects are modeled by a sequence of objects with a pseudo-periodic arrival rate.

(2) While packets from an object are sent, the object is active on the network until the packets from the present object are completely transmitted. After that, the object becomes inactive on the network. Then, based on the server's delivery queue, the next object can be transmitted.

(3) Network propagation delay can be seen as a constant for many types of network. It is so small that it can be neglected in the model. Hence, the network delay of an object is given by the total transmission time needed for the object.

Model:

- Network bandwidth available per object is given by network bandwidth / the number of objects active in network transmission – C, where C is an overhead due to multiple servers and can be changed in simulation.
- Transmission time is given by size of object / network bandwidth available per object.
- Network delay is given by size of object / network bandwidth available per object.

Obviously, there is a maximum value of the number of objects active in network transmission beyond which the network bandwidth available per object becomes negative, and that means the network is saturated.

Two events affect the network status. One is that a server sends an object to the network and the object becomes active. The other is that an active object completely finishes its transmission and becomes inactive. When one of the two events happens, the network bandwidth available per object changes dynamically because the number of objects active in network transmission is a parameter variable in time. Hence, our simulation model is a time and event driven dynamic model. Whichever event happens, the network needs to recompute the network bandwidth available per object and the transmission time because the event may cause two changes. One is the number of objects active in network transmission, and the other is the size of remaining parts of transmitted objects.

In the recomputation, first the size of object needs to be replaced by the size of remaining parts of the object: size of object – dt*network bandwidth available per object, where dt is the duration from the time last event happens until the time current event happens. Second, the network bandwidth available per object is recomputed with a new value of the number of objects active in network transmission. Finally, the new transmission time needs to be recomputed for each object.

6.1.5 Client Simulation Model

In our client simulation model, clients can send request to multiple servers over network to retrieve media objects at the same time to complete a multimedia presentation. We give some assumptions as follows.

(1) A client can receive multiple media objects at the same time at the client buffer, and transmit them to the presentation system.

(2) Clients have an enough throughput needed for the presentation of media objects to transmit them to the presentation system. Stream objects are modeled by a sequence of objects with a pseudo-periodic transmission rate.

Model:

- Client throughput available per object is given by a constant.
- Client delay time is given by size of object / client throughput available per object.

Then, after client delay time, transmitted media objects at clients can be presented by the presentation system.

6.2 Implementation in a Simulated Environment

6.2.1 Module Design

The scheduler includes three modules: the presentation scheduling unit, the delivery scheduling unit and the run time rescheduling unit. A server scheduler is needed in servers to cooperate with rescheduling. The simulation of a distributed multimedia

environment consists of three modules: the server simulator, the network simulator and the client simulator. The presentation simulator simulates the results of a multimedia presentation. FIGURE 25 shows the design.

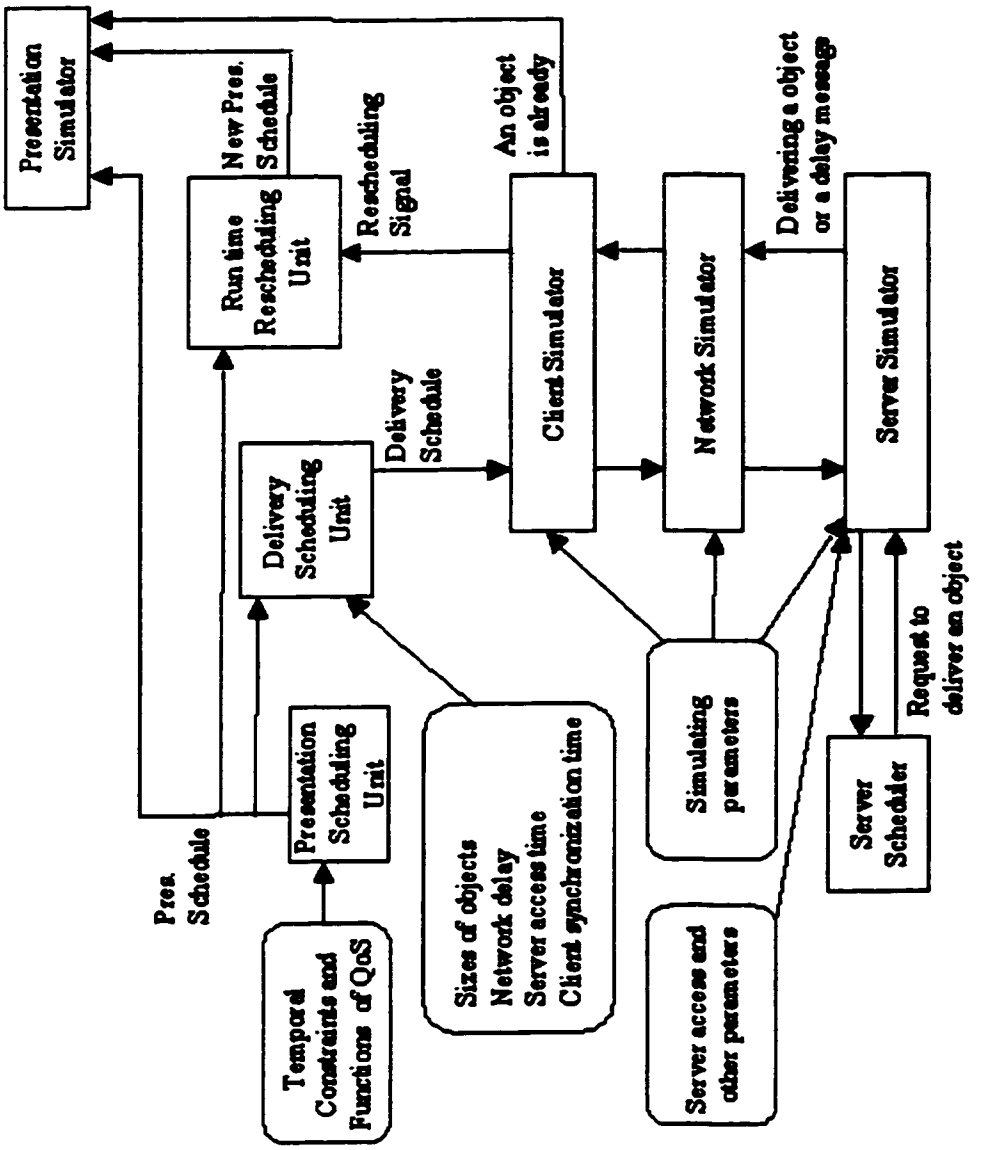


FIGURE 25: System design of a scheduler in a simulated distributed environment

6.2.2 Server Simulator

The functionality of the server simulator is to reflect the dynamic delivery of media objects in a distributed environment. It may send messages to network or call other modules upon receiving a message from network or the server scheduler to complete delivering objects to clients. Before putting data units on to network, a server needs to transfer data units into the buffer from secondary storage and package them. That causes a server delay in the server. The server delay is simulated by an initial total available throughput and traffic functions, and the duration of the server delay depends on the traffic of the server and the size of objects in the given server. One server can offer services to multiple clients.

A server needs to communicate with the network or with the client through network.

- Message from network: The messages from network to a server are periodic delivery schedules from clients.
- Message to network: There are two kinds of messages from a server to network. One is the delay message of an object to clients. The other is a delivering message that tells network that the server has put a specific object on to network.

Servers retrieve objects from the secondary storage to the server buffer and send them to the network. We use a producer-consumer model to analyze the transmission of objects from servers to network. Servers produce media objects to server buffers, and network consumes media objects from server buffers.

To simplify the implementation, we assume the producing rate of servers is greater than or equal to the consuming rate of network, and servers have sufficient

buffers to store objects. In other words, we need not cope with the underflow and overflow of server buffers.

As soon as an atomic object or a starting chunk for stream objects arrives at server buffers, a server sends it to network. That means the server may send multiple objects to network at the same time.

6.2.3 Network Simulator

The main functionality of the network simulator is to reflect the network's dynamic ability of transferring data on it. This ability is measured by the traffic or dynamically changed loading on network, an initial network bandwidth, and traffic functions. Hence, the delay duration depends on the traffic and the size of objects in a given network.

Network receives objects from servers and transmits them to clients. We use a producer-consumer model to analyze the transmission of objects from network to clients. Network produces media objects, and clients consume these objects.

To simplify the implementation, we assume the producing rate of network is greater than or equal to the consuming rate of clients, and clients have sufficient buffers to store objects. In other words, we need not cope with the underflow and overflow of client buffers.

Clients may receive multiple objects from network at the same time.

6.2.4 Client Simulator

The functionality of the client simulator is to reflect the dynamic retrieval of media objects in a distributed environment. It may call other modules or send messages to network upon receiving a message from network or user to complete a scheduling of a multimedia presentation. After arriving at a client, data units need to be transferred into the buffer, and that causes a buffer delay in client. This delay is measured by the client throughput as discussed. The duration of client delay depends on the client throughput and the size of objects.

A client needs to communicate with network or with servers through network.

- Message to network: The messages from a client to network are periodic delivery schedules to servers.
- Message from network: There are two kinds of messages from network to a client. If the client receives a delay message of an object from servers, it needs to reschedule the presentation. If the client receives an arrival message of a media object from network, it can present the object.

6.2.5 Server Scheduler

The server scheduler determines which object is serviced next. It adopts the earliest deadline first (EDF) algorithm to serve a task and considers delivery schedules as deadlines. Also, the scheduling is affected by the server access delay.

Actually, the server access delay includes a transfer delay in storage device and a network transport buffer delay in the server. To simplify, we combine them as one parameter, i.e. the throughput of the server access because this throughput is the bottleneck of the server access and can state the dynamic features of the server. Furthermore, the server access delay ds depends on the traffic of the server and the size of objects as discussed.

Because of the limitation of the server throughput, the server scheduler computes and compares the available throughput and needed throughput for delivering a specific media object before the object can be put onto the network. If the required throughput cannot be satisfied, the delivery of the object cannot be on schedule. In this case, the server should inform the client that the delivery of the object has to be delayed. Otherwise the object can be delivered on schedule.

6.2.6 Presentation Scheduling Unit

The presentation scheduling unit takes a temporal specification or an ICG as the input. It computes and outputs an optimal schedule for the presentation. Furthermore, the presentation schedule is sent to other modules to be carried out.

6.2.7 Delivery Scheduling Unit

The delivery-scheduling unit takes a presentation schedule, the size of objects, and the estimated values of network delay, server delay and client synchronization time as the input. It computes and outputs a delivery schedule for the presentation.

6.2.7.1 Periodic Delivery Schedule

Because of the dynamic behavior of a distributed environment, a delivery schedule may only be valid for a limited time. Hence, the delivery scheduling unit computes a part of delivery schedule periodically and sends it to servers. A new delivery schedule is created for a new period when the current schedule expires.

The duration of periods is chosen based on the implementation requirement. For instance, 10 minutes can be a choice.

6.2.7.2 Computation of a Delivery Schedule in Implementation

While clients present a multimedia document, a request from clients may experience different amounts of delays on the paths to servers. To simplify the implementation, we ignore the delays because we think the size of the request, such as a periodic delivery schedule, is very small and they can reach the servers as soon as possible. Hence the delays from the request are small enough to be ignored. On the other hand, we emphasize the transformation of media objects from servers to clients.

Delivery schedules determine the time instant at which media objects are put onto network from the network transport support in servers. As we discussed previously, delivery schedules can be derived from presentation schedules, the server delay, the

network delay, and the client process delay. The network delay includes the network access delay and the network transmission delay.

To simplify the problem, we assume the server access delay is proportional to the size of objects for atomic objects or chunk of frames for stream objects and is determined from a server throughput. The server access delay is given by size of object / server throughput for the object.

The network propagation delay or transmission delay can be seen as a constant for many types of network. For instance, it is often assumed as 100μs for ISDN which is small enough to be ignored in our implementation because the transformation of a media object needs much more time generally.

The network access delay is proportional to the size of objects for atomic objects or chunk of frames for stream objects. Furthermore, it is determined from the network bandwidth. The network access delay is given by size of object / bandwidth needed for the object. The needed bandwidth had better accord with the client throughput.

Obviously, the client process delay is client system dependent; hence it can be computed by the local system and transferred to the scheduling system. For instance, the client process delay with decoding and without decoding is computed differently. In our implementation, we assume the client process delay is proportional to the size of objects for atomic objects or chunk of frames for stream objects and determined from a client throughput. The client process delay is given by size of object / client throughput for the object.

Consequently, we can derive delivery schedules from presentation schedules and other parameters.

6.2.8 Run Time Rescheduling Unit

The rescheduling unit is responsible for recomputing a presentation schedule at run time that eliminates or minimizes inconsistencies. It is triggered when the network simulator sends a message that tells the client simulator that an object cannot be delivered on schedule, or the client simulator intends to present an object and finds that the object has not arrived.

We ran the rescheduling unit 1000 times on different graphs. Each of them took around 6 seconds; hence one run of the rescheduling unit takes 6 milliseconds cpu time on the average.

6.2.9 Presentation Simulator

The presentation simulator tries to simulate a presentation of multimedia documents on schedule. It takes presentation schedules and simulated messages from network as input. It outputs simulated information.

6.2.9.1 Outputs of Program

The execution of the scheduler in a simulated environment produces a series of outputs to report relevant information. The outputs include all events related to the scheduling and the simulation. We further use these outputs to analyze experiments and help us draw conclusions. Some events are as follows.

- The time of delivering an object from the server storage
- The time of receiving an object in the server buffer
- The time of receiving a starting chunk of stream objects in the server buffer
- The time of sending an object to network
- The time of receiving an object in clients
- The time of receiving a starting chunk of stream objects in clients
- The time of presenting an object to users

6.2.9.2 Deadline for Aborting a Presentation due to Inconsistent Objects

In order to present media objects, they need to be retrieved from servers to clients on schedule. During a presentation, media objects go through three main stages: server access, network transmission and client access. Hence they can be delayed with the server delay, the network delay and the client delay respectively.

These delays can cause inconsistencies to a multimedia presentation associated with temporal constraints in clients. To eliminate or reduce inconsistencies, the system tries to estimate the values of delays. However, clients cannot estimate these delays exactly because the server delay or network delay is independent of clients. Hence inconsistencies are unavoidable at run time. To solve this kind of inconsistency problems, clients need to decide a deadline to abort the presentation or endure inconsistencies. The users of the presentation will determine the deadline, for it should be acceptable to the users.

The deadline is determined in such a way that each interval constraint [a, b] in an ICG is given a threshold percentage p. Then the corresponding interval can be stretched

as [(1-p)a, (1+p)b]. For example, the default percentage is 30%; then the stretched interval for [10,20] is [7, 26]. A client will stop waiting for the arrival of an inconsistent object and aborts a presentation if the inconsistency exceeds the stretched interval.

This design can be used flexibly, and the size of the deadline expresses the hardness or softness of constraints. For instance, the percentage 0 expresses that a constraint is the hardest. In contrast, the default percentage 30% is soft; the percentage 50% is softer than the default percentage.

6.3 Experimental Analysis

Two categories of experiments are designed to analyze potential claims. One is to prove that QoS based model can lead to a better performance multimedia presentation. The other is to verify the adaptation of rescheduling.

In order to verify the above claims, we ran a series of temporal constraint graphs and compare the results among them. On one hand, we analyze the results from experiments by some important parameters and try to draw our conclusions. These important parameters are objective values, inconsistency values, the number of reschedulings, and presentation abortion status. On the other hand, we analyze the network load and try to get more evidences supporting our conclusions. We vary some parameters to observe the results, such as the bandwidth of the network, the throughput of servers and clients, the size of media objects, temporal constraints, and the deadline of aborting a presentation.

6.3.1 Experiment 1: Comparison between QoS Based Schedule and without QoS Schedule

The aim of this experiment is to study the performance of our QoS based model. For the purpose of comparing against schedulers without taking QoS into the model, we run the same experiments with a unity QoS on every temporal constraint. Then we can collect results for both a unity QoS and the actual QoS. Some experimental results regarding the comparison between QoS based schedule and with unity QoS schedule are shown in the following table, and more detailed results are in APPENDIX B.

		With QoS functions				With a Unity QoS			
No.	No.of Pres.	Objec. Value	incons. value	no.of resch.	Abort	Objec. Value	Incons. Value	no.of resch.	Abort
1	1	38	0	0	No	0	0	0	No
2	3	38	0	0	No	0	0	0	No
3	6	87	0	0	No	0	0	0	No
4	3	87	0	0	No	0	0	0	No
5	12	337	0	4	No	25	0	3	No
6	6	321	0	10	No	43	0	10	No
7	1	349	0	0	No	19	0	0	No
8	9	29	0	4	No	3	0	3	No
9	6	23	0	6	No	4	0	6	No
10	6	23	0	6	No	4	0	6	No
11	6	32	0	3	No	10	0	3	No
12	9	79	0	4	No	12	0	4	No
13	12	79	0	4	No	9	0	4	No
14	3	85	0	0	No	6	0	0	No

By comparing results between QoS functions-based and with unity QoS values in every run, we note that QoS functions-based schedule produces higher objective values in all cases. In other words, our QoS model can lead to better presentation performance. In

conclusion, QoS model can achieve better presentation performance than without QoS model.

By comparing all the results between QoS functions-based and with unity QoS values, there are no obvious differences regarding the need of rescheduling. In both cases, they give similar behaviors as we discussed in the comparison between with rescheduling and without rescheduling. For example, the need of rescheduling increases as the network load increases.

By comparing all the results, we note that there are no obvious differences between the small and large values of intervals. Also, there are no obvious differences between the small and large variation of objects. Moreover, topology and large graphs do not seem to lead to big differences.

6.3.2 Experiment 2: Comparison between Rescheduling and without Rescheduling

The aim of this experiment is to study the performance of rescheduling. We disable rescheduling and conduct experiments by letting the derived optimal presentation schedule to be followed without rescheduling. Hence failures of object deliveries due to network or server delays are simply ignored. Then we can collect results for both with rescheduling and without rescheduling for the same data set, i.e., temporal constraints and the rest of system parameters. Some experimental results regarding the comparison between rescheduling and without rescheduling are shown in the following table, and more detailed results are in APPENDIX B.

		With Rescheduling				Without Rescheduling			
No.	No.of Pres.	objec. Value	incons. value	no.of resch.	Abort	Objec. Value	incons. value	no.of resch.	Abort
1	9	32	0	2	No				Yes
2	6	1	1	103	No				Yes
3	12	75	0	4	No				Yes
4	6	32	0	28	No				Yes
5	12	337	0	4	No				Yes
6	6	321	0	10	No				Yes
7	1	349	0	0	No	349	0	0	No
8	6	35	0	1	No				Yes
9	6	23	0	6	No				Yes
10	6	23	0	6	No				Yes
11	6	32	0	3	No				Yes
12	9	79	0	4	No				Yes
13	9	83	0	2	No				Yes
14	3	85	0	0	No	85	0	0	No

By comparing results between with rescheduling and without rescheduling in every run, we note that the abortion of presentation without rescheduling happened earlier than with rescheduling. In conclusion, the rescheduling strategy is more effective than without rescheduling.

By comparing all the results with rescheduling in every run, we note that the results are same when the network load is small because no rescheduling occurs. That means rescheduling is not necessary when the network load is small. However, when the network load is high, rescheduling occurs and leads to fewer inconsistencies. Similar situation takes place without rescheduling. In conclusion, rescheduling outperforms under heavy network load. Also, the analysis of the network load provided evidences to the conclusion.

By comparing all the results between the small and large variations of intervals, we note that abortion of presentation happened earlier with small standard deviation than

with large standard deviation because the large variation of intervals gives more room to reschedule the temporal constraint graph. In other words, loose temporal constraints are easier to be rescheduled. In conclusion, having a large variation of intervals leads to better adaptability of rescheduling when everything else is fixed.

By comparing the results between small and large objects, we note that more rescheduling is needed and the abortion of presentation happened earlier for large objects because the network load is heavier than small objects. In conclusion, large objects lead to worse performance of rescheduling in a multimedia presentation than small objects when everything else is fixed.

By comparing all the results, we note that there are no obvious differences between the small and large values of intervals. Also, there are no obvious differences between the small and large variation of objects. Moreover, topology and large graphs do not seem to lead to big differences.

CHAPTER 7 Conclusions

The previous chapters described the design and implementation of a synchronization scheduler for multimedia presentations in a distributed environment.

CHAPTER 1 addressed the area of this thesis, i.e. the temporal synchronization of orchestrated multimedia presentation. CHAPTER 2 discussed some requirements of multimedia presentation in a distributed environment. CHAPTER 3 described the techniques to produce a static optimal presentation schedule based on the QoS model. CHAPTER 4 developed an approach to reschedule the presentation at run time based on retiming method. CHAPTER 5 introduced the design of the scheduler. To prove the usefulness and effectiveness of the scheduler, CHAPTER 6 described our implementation in a simulated distributed environment and analyzed some simulation experiments.

Based on the above work, we can give some conclusions. The scheduler developed in the thesis can derive an optimal presentation schedule which can lead to a multimedia presentation with better QoS performance, while traditional schedulers do not take into consideration QoS models of temporal constraints. Here "better" means that a multimedia document can be presented with better qualities of temporal constraints, or that leads to a better presentation. In addition, the run time rescheduling is more adaptive. As analyzed in SECTION 6.3 (Experimental Analysis), the rescheduling leads to fewer inconsistencies under heavy system load.

Our implementation is only in a simulated environment, and it is envisaged that more future work is possible. First, the estimation of delivery schedule is different for different networks and multimedia applications. Our implementation adopted a very simple model, hence needs more development in a real distributed environment. Second,

some strategies, such as how to decide to abort a presentation in the case of synchronization errors, still need to be considered more elaborately. Third, our implementation did not take care of the event from server to inform clients about synchronization error, so how to schedule tasks in server is a necessary work in a real environment. Finally, the most important part is how to cope with more complex QoS models than what we developed in this thesis, for example, QoS functions are not easily translated to linear forms.

BIBLIOGRAPHY

[BF98] E. Bertino, E. Ferrari. Temporal Synchronization Models for Multimedia Data. IEEE Transaction on Knowledge and Data Engineering, Vol.10, No.4, pp. 612-631, July/August 1998.

[BZ92] M.C. Buchanan, P.T. Zellweger. Specifying Temporal Behavior in Hypermedia Documents. In Proc. of the ACM European Conference on Hypertext, pp. 262-271, Milan, Italy, December 1992.

[BZ93] M.C. Buchanan, P.T. Zellweger. Automatic Temporal Layout Mechanisms. In Proc. of the First ACM International Conference on Multimedia, pp. 341-350, Anaheim, California, August 1993.

[CCG96] D. Chen, R. Colwell, H. Gelman, P.K. Chrysanthis, D. Mosse. A Framework for Experimenting with QoS for Multimedia Services, Technical Report, Department of Computer Science, University of Pittsburgh, Pennsylvania, 1996.

[CPS96] K.S. Candan, B. Prabhakaran, V.S. Subrahmanian. CHIMP: A Framwork for Supporting Distributed Multimedia Document Authoring and Presentation. In Fourth ACM International Multimedia Conference, Boston, Mass., pp. 329-340, 1996.

[CPS98] K.S. Candan, B. Prabhakaran, V.S. Subrahmanian. Retrieval Schedules Based on Resource Availability and Flexible Presentation Specifications. ACM / Springer Multimedia Systems, Vol.6, No.4, pp. 232-250, July 1998.

[DMP91] R. Dechter, I. Meiri, J. Pearl. Temporal Constraint networks, Artificial Intelligence, Vol. 49, pp. 61-95, 1991.

[F90] D. Ferrari. Client Requirements For Real-Time Communication Services, IEEE Communication Magazine, Vol. 28, No. 11, pp. 65-72, November 1990.

[F95] S.Flinn. Coordinating Heterogeneous Time-Based Media Between Independent Applications, ACM Multimedia Conference'95, pp. 435-444, 1995.

[GL96] J.F. Gibbon, T.D.C. Little. The Use of Network Delay Estimation for Multimedia Data Retrieval. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 7, pp. 1376-1387, September 1996.

[JG97] J.P. Jarmasz, N.D. Georganas. Designing a Distributed Multimedia Synchronization Scheduler. Proc. IEEE Multimedia Systems'97, Ottawa, June 1997.

[JLS97] M. Jourdan, N. Layaïda, L. Sabry-Ismail. Time Representation and Management in Madeus: an authoring environment for Multimedia documents, in Proceedings of Multimedia Computing and Networking 1997, SPIE 3020, SanJose, February 1997.

[JSN95] P.W. Jardetzky, C.J. Sreenan, R.M. Needham. Storage and Synchronization for Distributed Continuous Media. ACM / Springer Multimedia Systems, Vol.3, pp. 151-161, 1998.

[JZ97] T.V. Johnson, A. Zhang. A Framework for Supporting Quality-Based Presentation of Continuous Multimedia Streams, Technical Report,

Department of Computer Science, State University of New York at Buffalo, Buffalo, 1997.

[KS95] M.Y. Kim, J. Song. Multimedia Documents with Elastic Time, ACM Multimedia Conference'95, pp. 143-154, 1995.

[KSF91] D.D. Kandlur, K.G. Shin, D. Ferrari. Real-Time Communication in Multi-hop Networks, Technical Report, Department of EE&CS, University of California, Berkeley, 1991.

[L93] T.D.C. Little. A Framework for Synchronous Delivery of Time-Dependent Multimedia Data. Multimedia Systems, pp. 175-200, 1993.

[L96] G. Lu. Communication and Computing for Distributed Multimedia Systems, Artech House, 1996.

[LG91] T.D.C. Little, A. Ghafoor. Multimedia Synchronization Protocols for Broadband Integrated services, IEEE Journal on Selected Areas in Communications, Vol.9, No.9, pp. 1368-1382, December 1991.

[LG92] T.D.C. Little, A. Ghafoor. Scheduling of Bandwidth-Constrained Multimedia Traffic. Computer Communications, Vol.15, No.6, pp. 381-387, July/August. 1992.

[LG93] T.D.C. Little, A. Ghafoor. Interval-Based Conceptual Models for Time-Dependent Multimedia Data. IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.4, pp. 551-563, August 1993.

[LLG94] L. Lamont, L. Li, and N.D. Georganas. Centralized and Distributed Architectures for Multimedia Presentational Applications. In Broadband Islands'94, Connecting with the End-User, Proceedings of the 3rd International

Conference on Broadband Islands, Hamburg, Germany, 7-9 June, pp. 59-70, Elsevier Science B.V., Amsterdam, The Netherlands, 1994.

[NS96] S.G. Nash and A. Sofer. Linear and Nonlinear Programming, The McGraw-Hill, 1996.

[PL96] M.J. Perez-Luque and T.D.C. Little, A Temporal Reference Framework for Multimedia Synchronization, IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication), Vol. 14, No. 1, pp. 36-51, January 1996.

[PPY96] P. Pusopa, C. Pope, J. Yantchev. Real-Time Communication in ATM Networks, Technical Report, Department of Computer Science, University of Adelaide, Australia, 1996.

[RH95] K. Rothermel, T. Helbig. An Adaptive Stream Synchronization Protocol, Technical Report, University of Stuttgart, Germany, 1995.

[RPT94] S.V. Raghavan, B. Prabhakaran and S.K. Tripathi. Synchronization Representation and Traffic Source Modeling in Orchestrated Presentation, Special issue on Multimedia Synchronization, IEEE Journal on Selected Areas in Communication, January 1995.

[RVR92] P.Venkat Rangan, Harrick M.Vin, and Srinivas Ramanathan. Designing an On- Demand Multimedia Service. IEEE Communication Magazine, pp.56-64, July 1992.

[S90] R. Steinmetz. Synchronization Properties in Multimedia Systems, IEEE J. on Selected Areas of Communication, Vol. 8, No. 3, pp. 401-412, April 1990.

[SN95] R. Steinmetz, K. Nahrstedt. Multimedia: Computing, Communications and Applications, Prentice Hall, 1995.

[V96] R.J. Vanderbei. Linear Programming : Foundations and Extensions. Kluwer Academic Publishers, 1996.

[WR93] T. Wahl, K. Rothermel. Representing Time in Multimedia-Systems, Technical Report, University of Stuttgart, Germany, 1993.

APPENDIX A: Format and Explanations of Input File

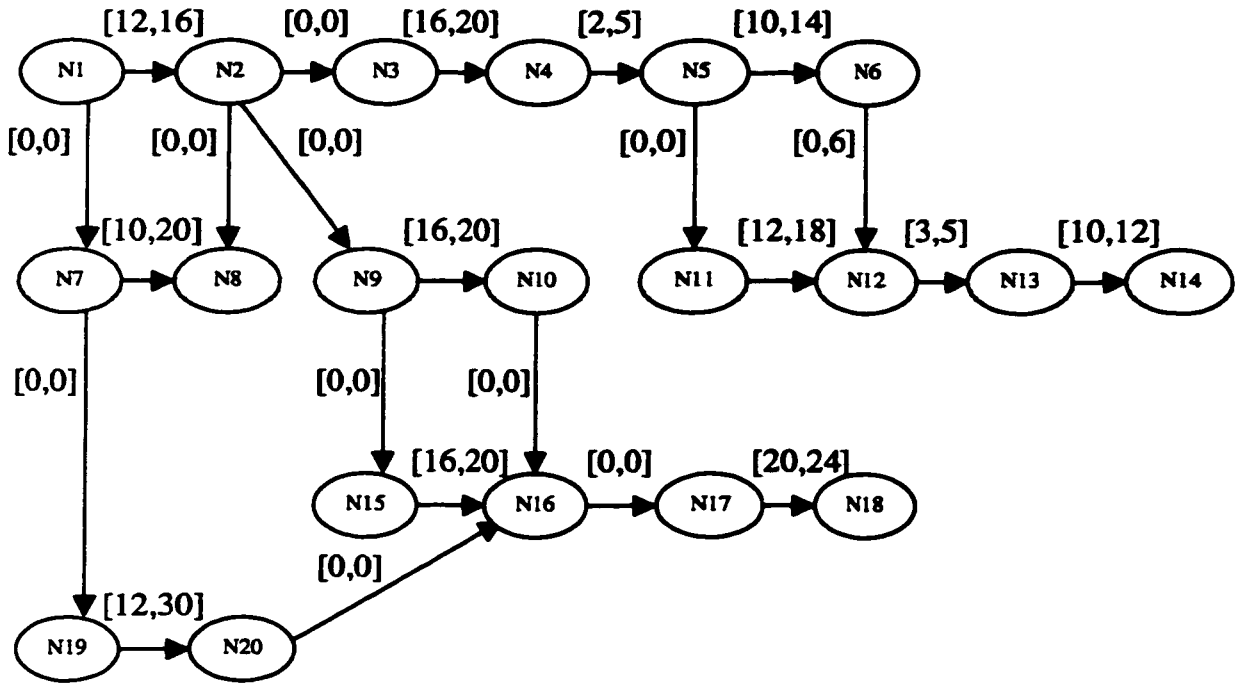


FIGURE 26: An Interval Constraint Graph

An interval constraint graph is shown in FIGURE 26. The graph has 20 nodes (events) N1, N2, ...N20 and some temporal constraints between events. Also, there are QoS functions between the start and end events of an object. We use the following to describe such an interval constraint graph.

* a temporal constraint graph

C1 | 0

C2 | -30%[12,16] | 0

C3 | 0 | -[0,0] | 0

C4 | 0 | 0 | -50%[16,20] | 0

C5 | 0 | 0 | 0 | -[2,5] | 0

C6 | 0 | 0 | 0 | 0 | -[10,14] | 0

C7 | -[0,0] | 0 | 0 | 0 | 0 | 0 | 0

C8 | 0 | -[0,0] | 0 | 0 | 0 | 0 | -%[10,20] | 0

C9 | 0 | -[0,0] | 0 | 0 | 0 | 0 | 0 | 0 | 0

C10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[16,20] | 0

C11 | 0 | 0 | 0 | 0 | 0 | -[0,0] | 0 | 0 | 0 | 0 | 0 | 0

C12 | 0 | 0 | 0 | 0 | 0 | 0 | -[0,6] | 0 | 0 | 0 | 0 | -[12,18] | 0

C13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[3,5] | 0

C14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[10,12] | 0

C15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[0,0] | 0 | 0 | 0 | 0 | 0 | 0

C16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[16,20] | 0

C17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[0,0] | 0

C18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[20,24] | 0

C19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -[0,0] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

C20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -20%[12,30] | 0

* Qos functions

F1 | 1 | 2 | 1 | -12 | -1 | 16

F3 | 3 | 4 | 1 | -16 | -1 | 20

F5 | 5 | 6 | 3 | -30 | -3 | 42

F7 | 7 | 8 | 3 | -30 | -3 | 60

F9 | 9 | 10 | 1 | -16 | -1 | 20

F11 | 11 | 12 | 3 | -36 | -3 | 54

F13 | 13 | 14 | 1 | -10 | -1 | 12

F15 | 15 | 16 | 3 | -48 | -3 | 60

F17 | 17 | 18 | 1 | -20 | -1 | 24

F19 | 19 | 20 | 3 | -36 | -3 | 90

* Size and type of objects

S1 | 1 | 2 | 100 | 100 | T

S2 | 3 | 4 | 9000 | 9000 | I

S3 | 5 | 6 | 10000 | 2000 | V

S4 | 7 | 8 | 6000 |1000 | A

S5 | 9 | 10 | 300 | 300 | T

S6 | 11 | 12 | 4000 | 100 | A

S7 | 13 | 14 | 600 | 600 | T

S8 | 15 | 16 | 6000 | 500 | V

S9 | 17 | 18 | 100 | 100 |T

S10 | 19 | 20 | 30000 | 1000 | V

The first part starting with character "C" is a matrix to describe the graph using the character "|" to separate different fields. If there is no edge between ith and jth node, then, there is a "0" in ith row and jth column of the matrix. Otherwise, there is a temporal constraint which is represented by the form: c%[a,b], where a and b are the lower bound

and upper bound of the interval respectively, c% represents the threshold (deadline) for aborting the presentation. Also, if there is a character "-" in the front of the above form: c%[a,b], that means the edge is from jth node to ith node. Otherwise, the edge is from ith node to jth node. For example, "C2 | -30%[12,16] | 0" represents the temporal constraint interval from node N1 to node N2 is [12, 16], and the threshold for aborting the presentation is 30%. We note that only the half of the matrix is enough to describe the graph, i.e., the left-down or right-up half. We prefer the lower left half.

The second part starting with character "F" is to describe QoS functions. Each line includes 6 fields separated by the character "|". The first number i and second number j represent that the QoS functions belong to the interval constraint in ith row and jth column in the matrix. Other fields represent the four coefficients of two QoS functions respectively. For instance, "F1 | 1 | 2 | 1 | -12 | -1 | 16" represents the QoS function between N1 and N2 are $x - 12$ and $-x + 16$.

The third part starting with character "S" is to describe the size and the type of a media object. Each line includes 5 fields separated by the character "|". The first number i and second number j represent that the object starting from ith node and ending at jth node is described. The third field represents the size of the object. The fourth field represents the size of the first segment of the object for video and audio objects (stream objects), and it is same as the third field for text and image objects (atomic objects). The fifth field which is one of the characters "T", "V", "A", and "I" represents the type of the object. Media types are classified as:

Text objects are represented by "T".

Video objects are represented by "V".

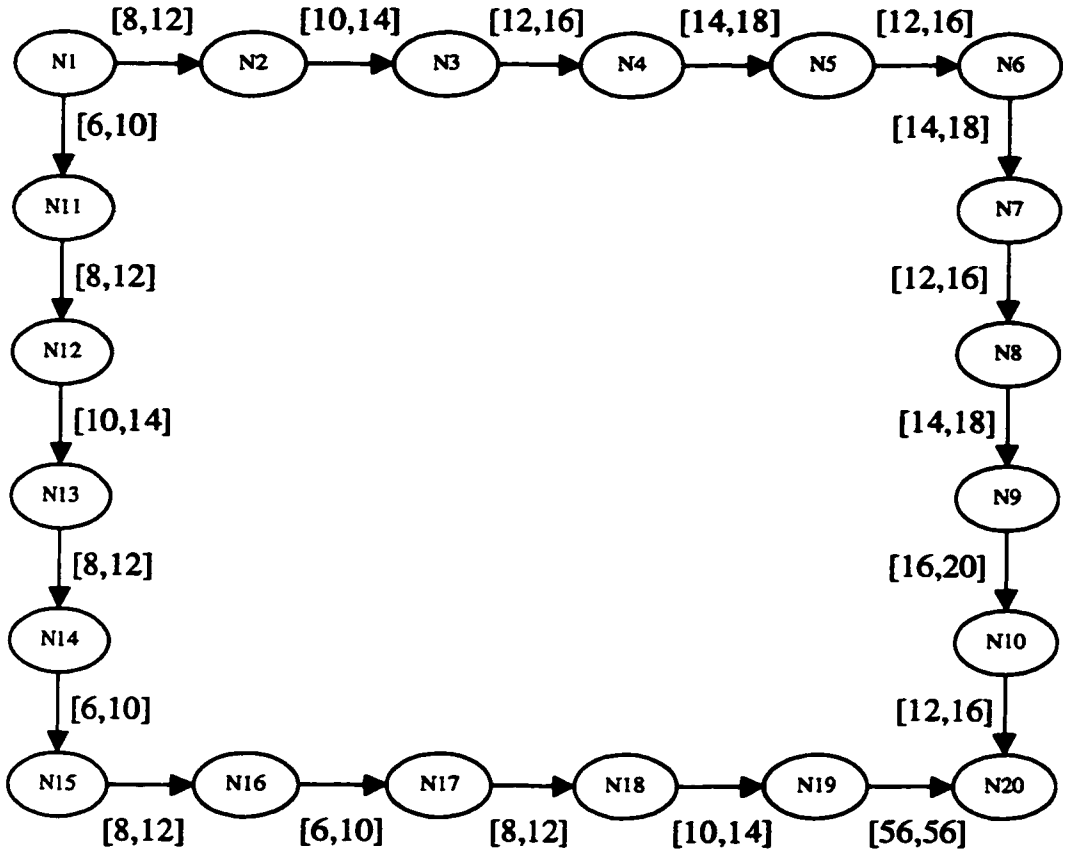
Audio objects are represented by "A".

Image objects are represented by "I".

For example, "S3 | 5 | 6 | 10000 | 2000 | V" represents object 3, which starts from N5 and ends to N6, is a video object. The size of object is 10000, and the size of first segment that must arrive at clients before a presentation is 2000.

APPENDIX B: Experimental Results

1. Small values and small S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	200	200	Text
2	3	4	9000	1500	Audio
3	5	6	450	450	Text
4	7	8	225	225	Text
5	9	10	3375	562	Audio
6	11	12	5061	843	Audio
7	13	14	6000	1000	Video
8	15	16	300	300	Text
9	17	18	4500	750	Audio
10	19	20	6750	1125	Video

QoS function for the constraint between N1 and N2: x - 8 and 12 - x

QoS function for the constraint between N3 and N4: 3x - 36 and 48 - 3x

QoS function for the constraint between N5 and N6: x - 12 and 16 - x

QoS function for the constraint between N7 and N8: x - 12 and 16 - x

QoS function for the constraint between N9 and N10: 3x - 48 and 60 - 3x

QoS function for the constraint between N11 and N12: 3x - 24 and 36 - 3x

QoS function for the constraint between N13 and N14: 3x - 24 and 36 - 3x

QoS function for the constraint between N15 and N16: x - 8 and 12 - x

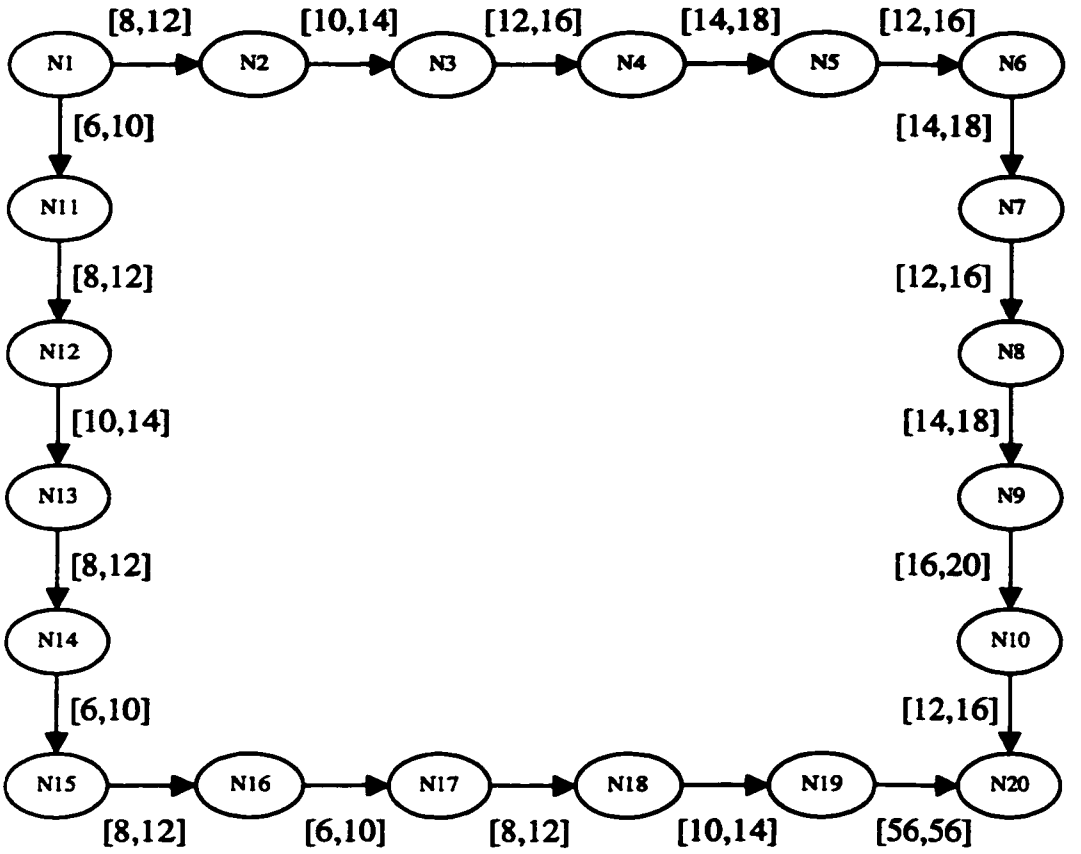
QoS function for the constraint between N17 and N18: 3x - 24 and 36 - 3x

QoS function for the constraint between N19 and N20: 3x - 168 and 168 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. Value	incons. value	no.of resch.	Abort	Objec. Value	incons. value	no.of resch.	Abort
1	38	0	0	No	38	0	0	No
3	38	0	0	No	38	0	0	No
6	38	0	0	No	38	0	0	No
9	32	0	2	No				Yes
12	22	0	7	No				
15	11	0	10	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	Objec. Value	Incons. Value	no.of resch.	Abort
1	38	0	0	No	0	0	0	No
3	38	0	0	No	0	0	0	No
6	38	0	0	No	0	0	0	No
9	32	0	2	No	6	0	2	No
12	22	0	7	No	12	0	6	No
15	11	0	10	No	12	0	10	No

2. Small values and small S.D. of intervals, large values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	2000	2000	Text
2	3	4	90000	15000	Audio
3	5	6	4500	4500	Text
4	7	8	2250	2250	Text
5	9	10	33750	5620	Audio
6	11	12	50610	8430	Audio
7	13	14	60000	10000	Video
8	15	16	3000	3000	Text
9	17	18	45000	7500	Audio
10	19	20	67500	11250	Video

QoS function for the constraint between N1 and N2: x - 8 and 12 - x

QoS function for the constraint between N11 and N12: 3x - 24 and 36 - 3x

QoS function for the constraint between N13 and N14: 3x - 24 and 36 - 3x

QoS function for the constraint between N15 and N16: x - 8 and 12 - x

QoS function for the constraint between N17 and N18: 3x - 24 and 36 - 3x

QoS function for the constraint between N19 and N20: 3x - 168 and 168 - 3x

QoS function for the constraint between N3 and N4: 3x - 36 and 48 - 3x

QoS function for the constraint between N5 and N6: x - 12 and 16 - x

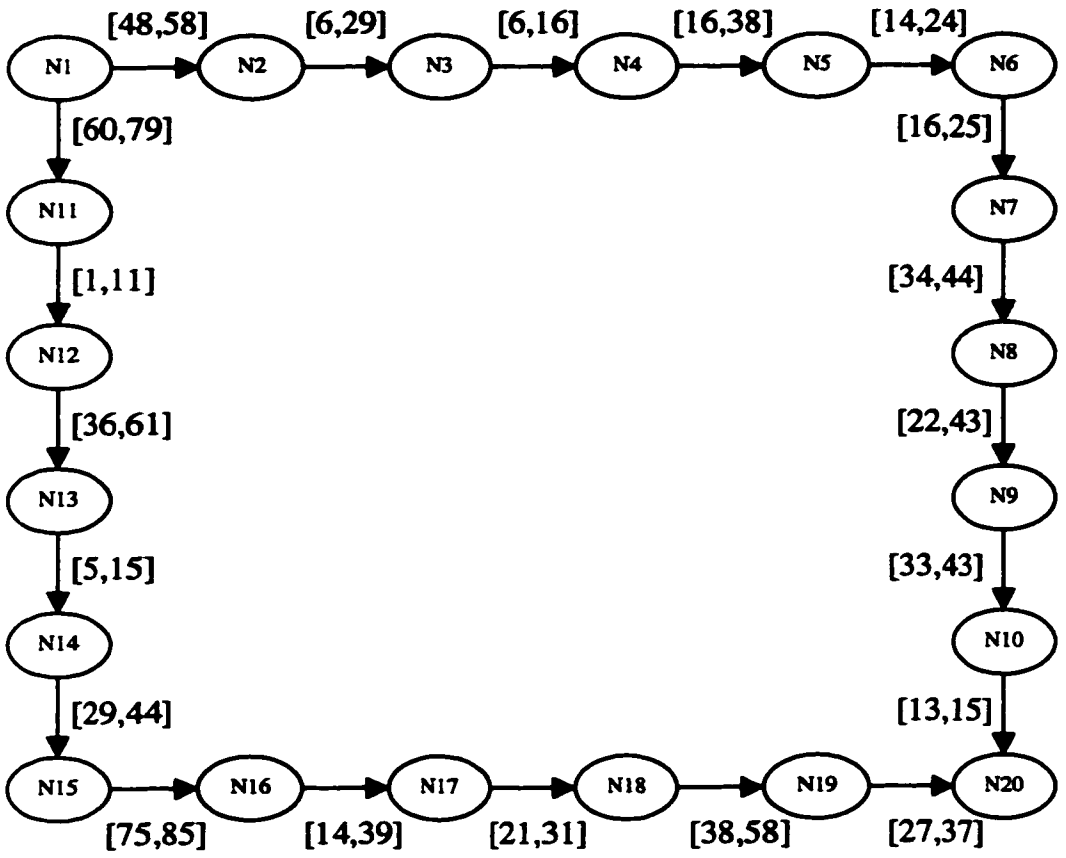
QoS function for the constraint between N7 and N8: x - 12 and 16 - x

QoS function for the constraint between N9 and N10: 3x - 48 and 60 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. Value	no.of resch.	Abort
1	38	0	0	No	38	0	0	No
3	38	0	0	No	38	0	0	No
6	1	1	103	No				Yes
9				Yes				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	38	0	0	No	0	0	0	No
3	38	0	0	No	0	0	0	No
6	1	1	103	No	4	1	101	No
9				Yes				Yes

3. Small values and large S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	200	200	Text
2	3	4	9000	1500	Audio
3	5	6	450	450	Text
4	7	8	225	225	Text
5	9	10	3375	562	Audio
6	11	12	5061	843	Audio
7	13	14	6000	1000	Video
8	15	16	300	300	Text
9	17	18	4500	750	Audio
10	19	20	6750	1125	Video

QoS function for the constraint between N1 and N2: x - 48 and 58 - x

QoS function for the constraint between N3 and N4: 3x - 18 and 48 - 3x

QoS function for the constraint between N5 and N6: x - 14 and 24- x

QoS function for the constraint between N7 and N8: x - 34 and 44 - x

QoS function for the constraint between N9 and N10: 3x - 99 and 129 - 3x

QoS function for the constraint between N11 and N12: 3x - 3 and 33 - 3x

QoS function for the constraint between N13 and N14: 3x - 15 and 45 - 3x

QoS function for the constraint between N15 and N16: x - 75 and 85 - x

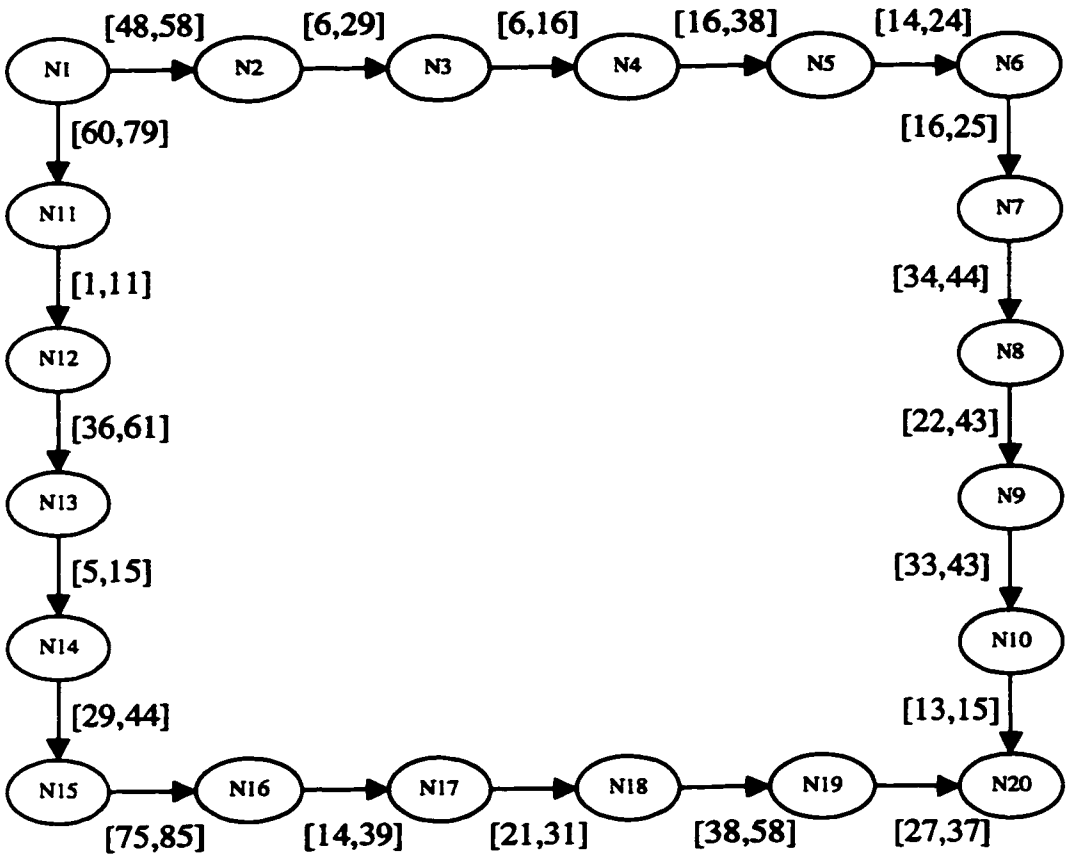
QoS function for the constraint between N17 and N18: 3x - 63 and 93 - 3x

QoS function for the constraint between N19 and N20: 3x - 81 and 111 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	87	0	0	No	87	0	0	No
3	87	0	0	No	87	0	0	No
6	87	0	0	No	87	0	0	No
9	87	0	0	No	87	0	0	No
12	75	0	4	No				Yes
15	69	0	6	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	87	0	0	No	0	0	0	No
3	87	0	0	No	0	0	0	No
6	87	0	0	No	0	0	0	No
9	87	0	0	No	0	0	0	No
12	75	0	4	No	6	1	4	No
15	69	0	6	No	9	1	6	No

4. Small values and large S.D of intervals, large values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	2000	2000	Text
2	3	4	90000	15000	Audio
3	5	6	4500	4500	Text
4	7	8	2250	2250	Text
5	9	10	33750	5620	Audio
6	11	12	50610	8430	Audio
7	13	14	60000	10000	Video
8	15	16	3000	3000	Text
9	17	18	45000	7500	Audio
10	19	20	67500	11250	Video

QoS function for the constraint between N1 and N2: x - 48 and 58 - x

QoS function for the constraint between N3 and N4: 3x - 18 and 48 - 3x

QoS function for the constraint between N5 and N6: x - 14 and 24- x

QoS function for the constraint between N7 and N8: x - 34 and 44 - x

QoS function for the constraint between N9 and N10: 3x - 99 and 129 - 3x

QoS function for the constraint between N11 and N12: 3x - 3 and 33 - 3x

QoS function for the constraint between N13 and N14: 3x - 15 and 45 - 3x

QoS function for the constraint between N15 and N16: x - 75 and 85 - x

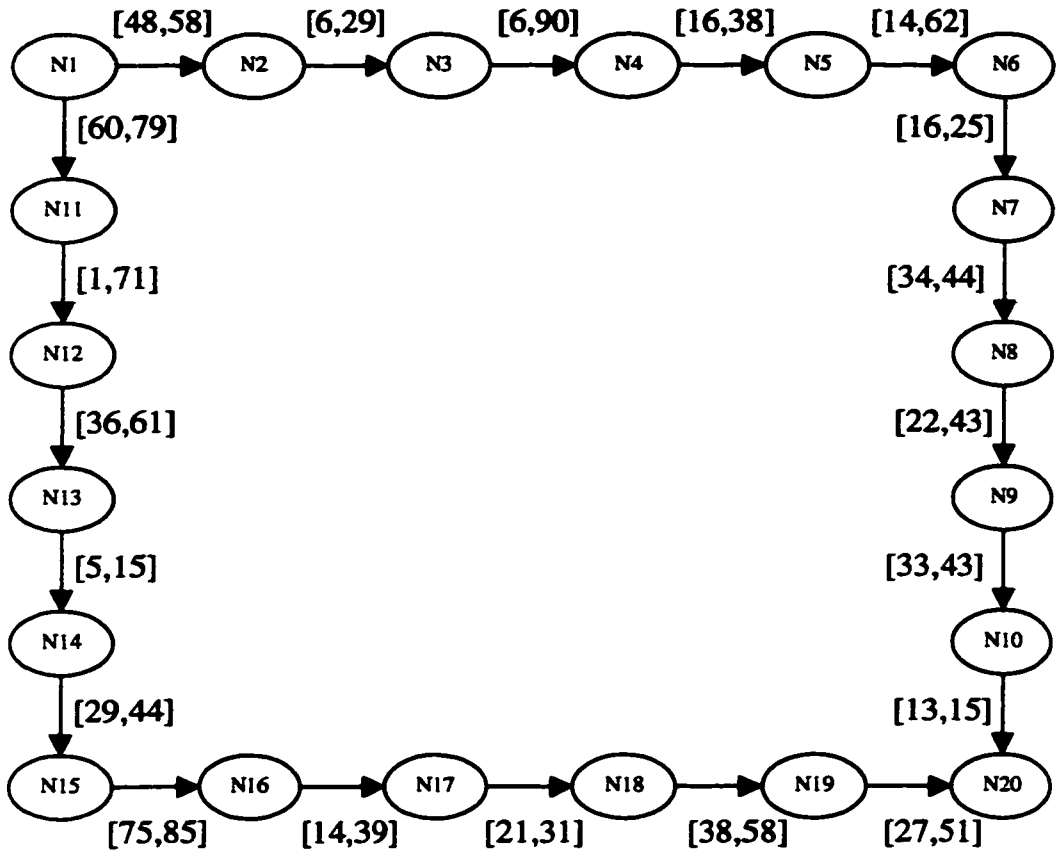
QoS function for the constraint between N17 and N18: 3x - 63 and 93 - 3x

QoS function for the constraint between N19 and N20: 3x - 81 and 111 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	Incons. value	no.of resch.	Abort
1	87	0	0	No	87	0	0	No
3	87	0	0	No	87	0	0	No
6	32	0	28	No				Yes
9	19	12	130	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	87	0	0	No	0	0	0	No
3	87	0	0	No	0	0	0	No
6	32	0	28	No	26	0	18	No
9	19	12	130	No	2	20	123	No

5. Small values and large S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	200	200	Text
2	3	4	9000	1500	Audio
3	5	6	450	450	Text
4	7	8	225	225	Text
5	9	10	3375	562	Audio
6	11	12	5061	843	Audio
7	13	14	6000	1000	Video
8	15	16	300	300	Text
9	17	18	4500	750	Audio
10	19	20	6750	1125	Video

QoS function for the constraint between N1 and N2: x - 48 and 58 - x

QoS function for the constraint between N3 and N4: 3x - 18 and 270 - 3x

QoS function for the constraint between N5 and N6: x - 14 and 62- x

QoS function for the constraint between N7 and N8: x - 34 and 44 - x

QoS function for the constraint between N9 and N10: 3x - 99 and 129 - 3x

QoS function for the constraint between N11 and N12: 3x - 3 and 213 - 3x

QoS function for the constraint between N13 and N14: 3x - 15 and 45 - 3x

QoS function for the constraint between N15 and N16: x - 75 and 85 - x

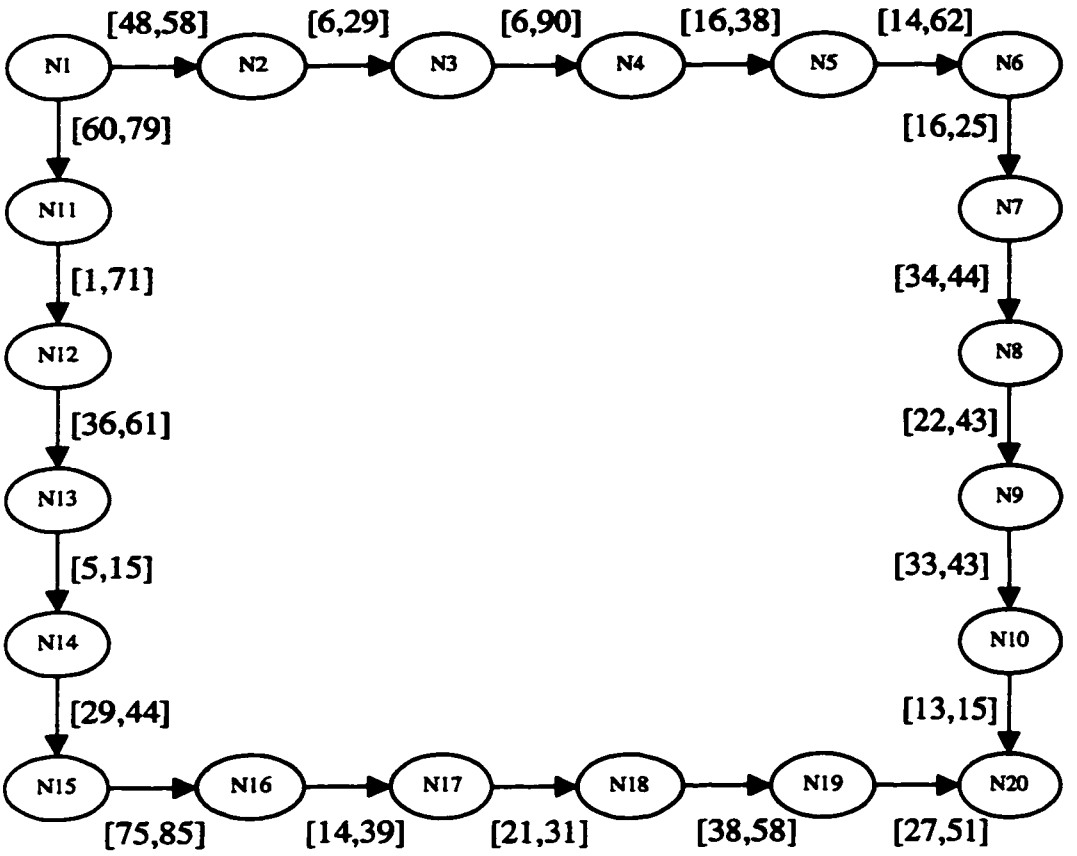
QoS function for the constraint between N17 and N18: 3x - 63 and 93 - 3x

QoS function for the constraint between N19 and N20: 3x - 81 and 153 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	349	0	0	No
3	349	0	0	No	349	0	0	No
6	349	0	0	No	349	0	0	No
9	349	0	0	No	349	0	0	No
12	337	0	4	No				Yes
15	331	0	6	No				
85	313	0	12	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	19	0	0	No
3	349	0	0	No	19	0	0	No
6	349	0	0	No	19	0	0	No
9	349	0	0	No	19	0	0	No
12	337	0	4	No	25	0	3	No
15	331	0	6	No	28	0	5	No
18	313	0	12	No	34	0	9	No

6. Small values and large S.D. of intervals, large values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	2000	2000	Text
2	3	4	90000	15000	Audio
3	5	6	4500	4500	Text
4	7	8	2250	2250	Text
5	9	10	33750	5620	Audio
6	11	12	50610	8430	Audio
7	13	14	60000	10000	Video
8	15	16	3000	3000	Text
9	17	18	45000	7500	Audio
10	19	20	67500	11250	Video

QoS function for the constraint between N1 and N2: x - 48 and 58 - x

QoS function for the constraint between N3 and N4: 3x - 18 and 270 - 3x

QoS function for the constraint between N5 and N6: x - 14 and 62- x

QoS function for the constraint between N7 and N8: x - 34 and 44 - x

QoS function for the constraint between N9 and N10: 3x - 99 and 129 - 3x

QoS function for the constraint between N11 and N12: 3x - 3 and 213 - 3x

QoS function for the constraint between N13 and N14: 3x - 15 and 45 - 3x

QoS function for the constraint between N15 and N16: x - 75 and 85 - x

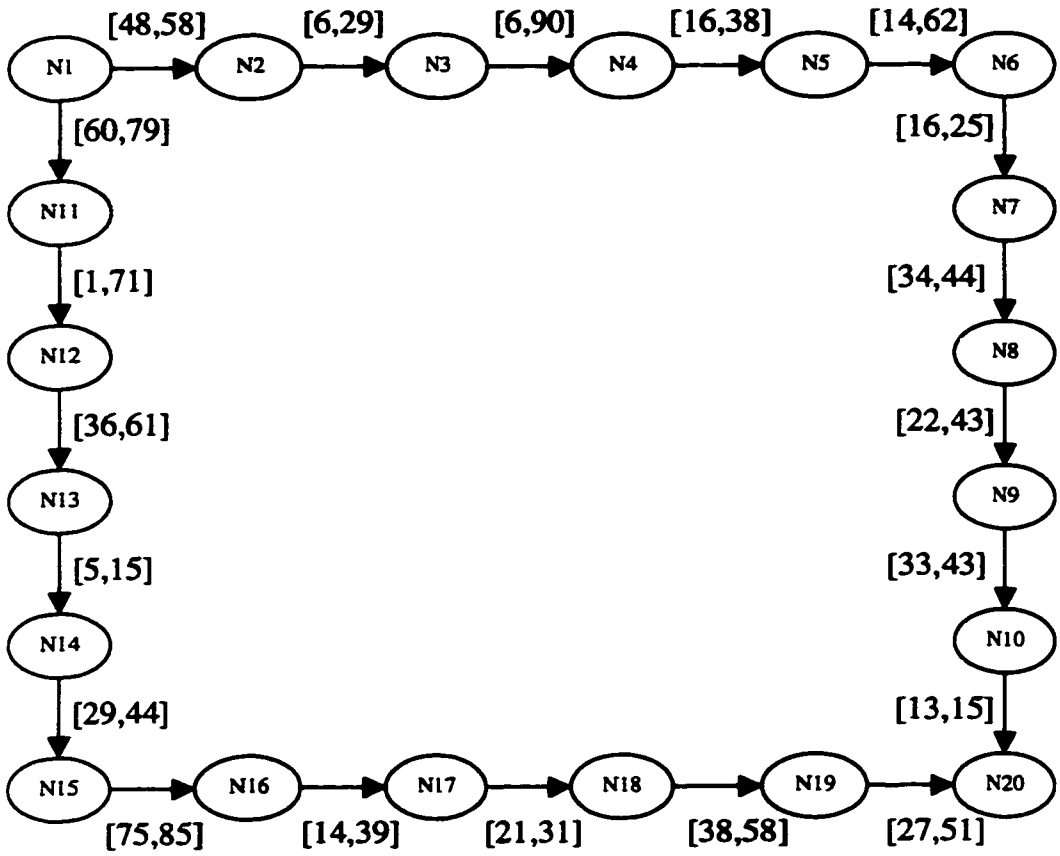
QoS function for the constraint between N17 and N18: 3x - 63 and 93 - 3x

QoS function for the constraint between N19 and N20: 3x - 81 and 153 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	349	0	0	No
3	349	0	0	No	349	0	0	No
6	321	0	10	No				Yes
9	168	0	120	No				
12	138	17	318	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	19	0	0	No
3	349	0	0	No	19	0	0	No
6	321	0	10	No	43	0	10	No
9	168	0	120	No	94	4	59	No
12	138	17	318	No	118	12	315	No

7. Small values and large S.D. of intervals, very large values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	20000	20000	Text
2	3	4	900000	150000	Audio
3	5	6	45000	45000	Text
4	7	8	22500	22500	Text
5	9	10	337500	56200	Audio
6	11	12	506100	84300	Audio
7	13	14	600000	100000	Video
8	15	16	30000	30000	Text
9	17	18	450000	75000	Audio
10	19	20	675000	112500	Video

QoS function for the constraint between N1 and N2: x - 48 and 58 - x

QoS function for the constraint between N3 and N4: 3x - 18 and 270 - 3x

QoS function for the constraint between N5 and N6: x - 14 and 62- x

QoS function for the constraint between N7 and N8: x - 34 and 44 - x

QoS function for the constraint between N9 and N10: 3x - 99 and 129 - 3x

QoS function for the constraint between N11 and N12: 3x - 3 and 213 - 3x

QoS function for the constraint between N13 and N14: 3x - 15 and 45 - 3x

QoS function for the constraint between N15 and N16: x - 75 and 85 - x

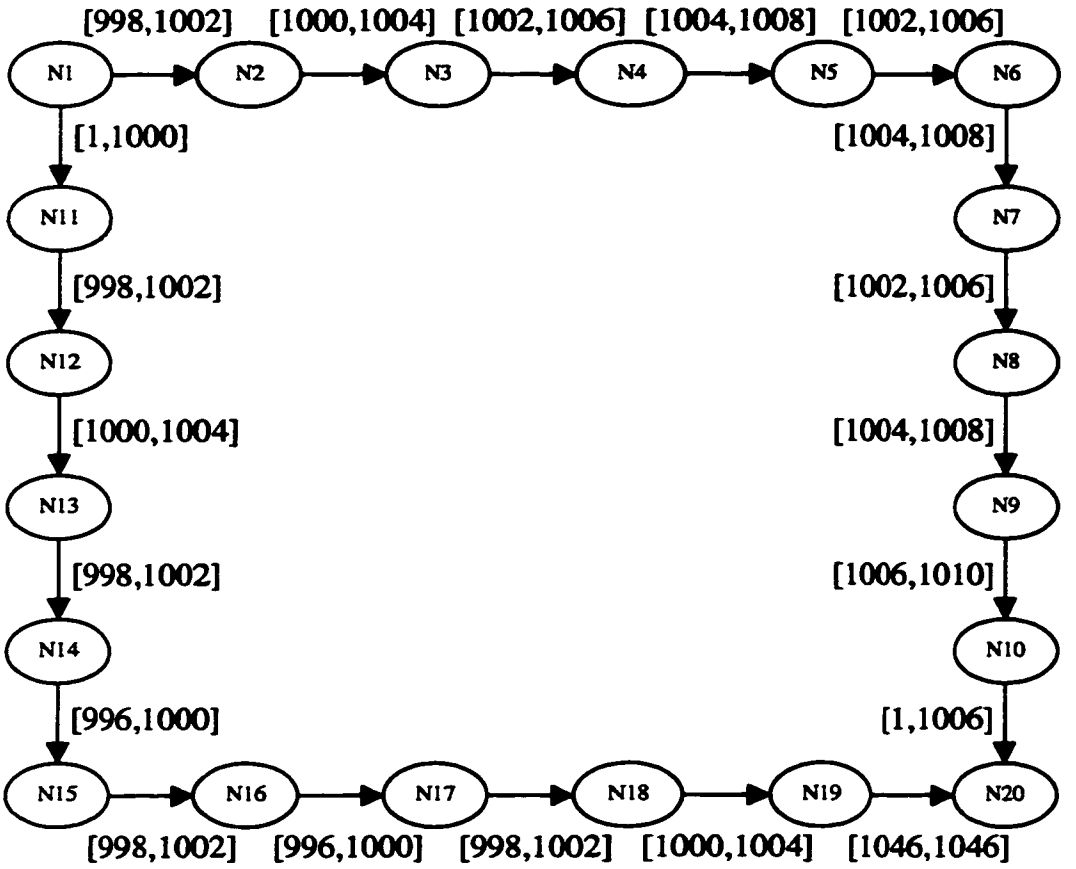
QoS function for the constraint between N17 and N18: 3x - 63 and 93 - 3x

QoS function for the constraint between N19 and N20: 3x - 81 and 153 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	Incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	349	0	0	No
3				Yes				Yes
6								
9								
12								
15								

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	349	0	0	No	19	0	0	No
3				Yes				Yes
6								
9								
12								
15								

8. Large values and small S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	200	200	Text
2	3	4	9000	1500	Audio
3	5	6	450	450	Text
4	7	8	225	225	Text
5	9	10	3375	562	Audio
6	11	12	5061	843	Audio
7	13	14	6000	1000	Video
8	15	16	300	300	Text
9	17	18	4500	750	Audio
10	19	20	6750	1125	Video

QoS function for the constraint between N1 and N2: x - 998 and 1002 - x

QoS function for the constraint between N3 and N4: 3x – 3006 and 3018 - 3x

QoS function for the constraint between N5 and N6: x - 1002 and 1006- x

QoS function for the constraint between N7 and N8: x - 1002 and 1006 - x

QoS function for the constraint between N9 and N10: 3x - 3018 and 3030 - 3x

QoS function for the constraint between N11 and N12: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N13 and N14: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N15 and N16: x - 998 and 1002 - x

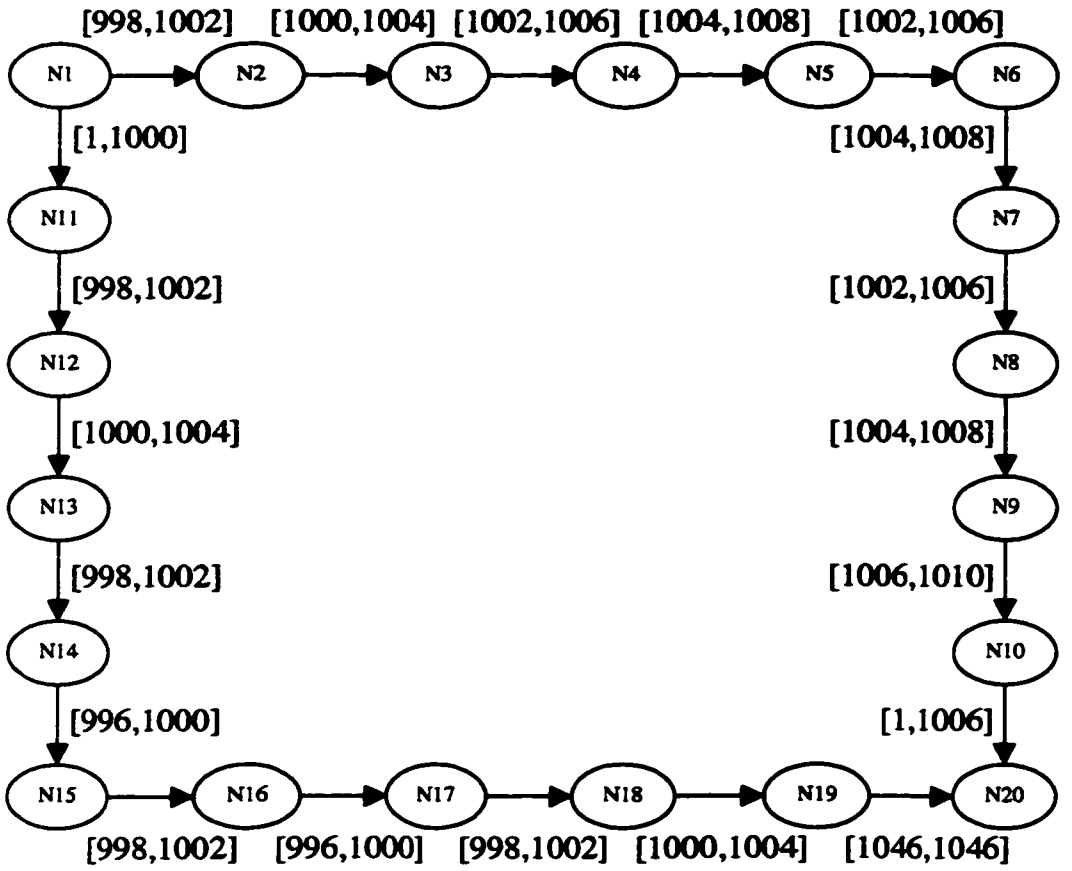
QoS function for the constraint between N17 and N18: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N19 and N20: 3x - 3138 and 3138 - 3x

	With Rescheduling				Without Rescheduling			
No.of Pres.	Objec. Value	incons. value	no.of resch.	Abort	objec. Value	incons. value	no.of resch.	Abort
1	38	0	0	No	38	0	0	No
3	38	0	0	No	38	0	0	No
6	35	0	1	No				Yes
9	29	0	4	No				
12	20	0	8	No				
15	12	0	14	No				

	With QoS functions				With a Unity QoS			
No.of Pres.	Objec. Value	incons. value	no.of resch.	Abort	objec. Value	incons. value	no.of resch.	Abort
1	38	0	0	No	0	0	0	No
3	38	0	0	No	0	0	0	No
6	35	0	1	No	3	0	1	No
9	29	0	4	No	3	0	3	No
12	20	0	8	No	9	0	7	No
15	12	0	14	No	12	0	10	No

9. Large values and small S.D. of intervals, large values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	2000	2000	Text
2	3	4	90000	15000	Audio
3	5	6	4500	4500	Text
4	7	8	2250	2250	Text
5	9	10	33750	5620	Audio
6	11	12	50610	8430	Audio
7	13	14	60000	10000	Video
8	15	16	3000	3000	Text
9	17	18	45000	7500	Audio
10	19	20	67500	11250	Video

QoS function for the constraint between N1 and N2: x - 998 and 1002 - x

QoS function for the constraint between N3 and N4: 3x – 3006 and 3018 - 3x

QoS function for the constraint between N5 and N6: x - 1002 and 1006- x

QoS function for the constraint between N7 and N8: x - 1002 and 1006 - x

QoS function for the constraint between N9 and N10: 3x - 3018 and 3030 - 3x

QoS function for the constraint between N11 and N12: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N13 and N14: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N15 and N16: x - 998 and 1002 - x

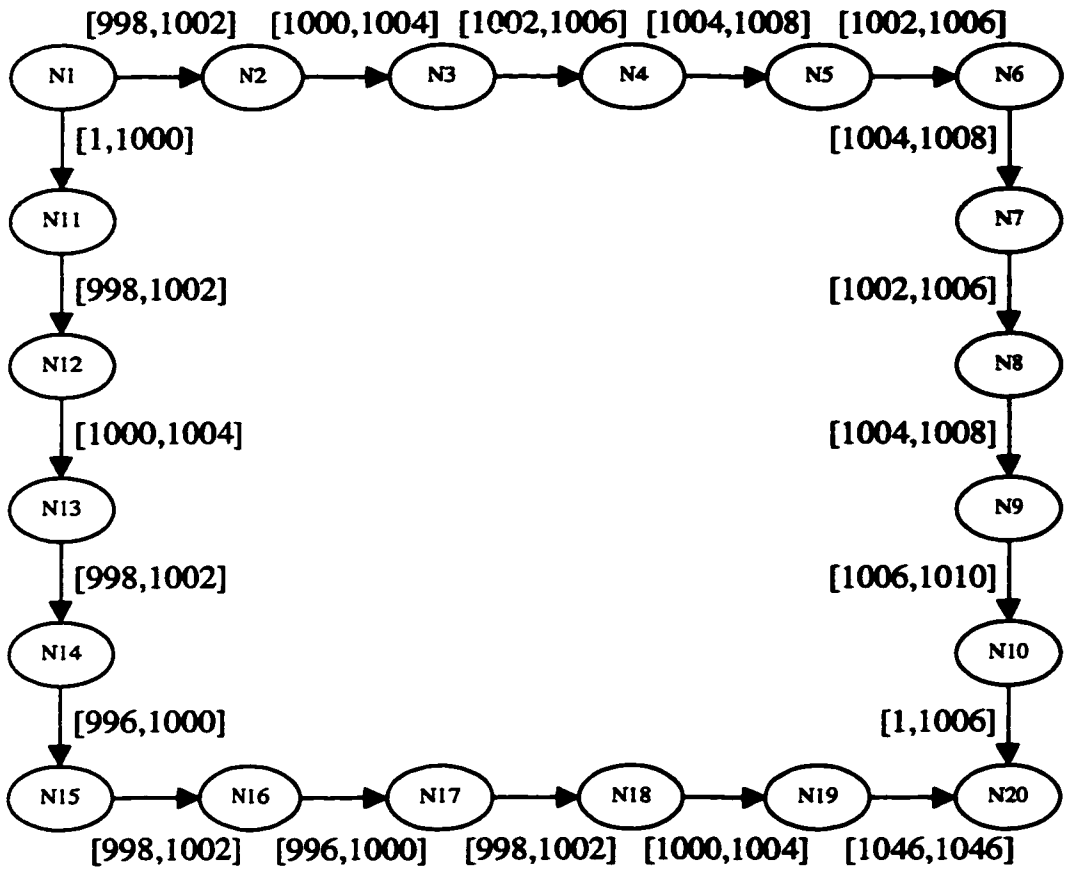
QoS function for the constraint between N17 and N18: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N19 and N20: 3x - 3138 and 3138 - 3x

	With Rescheduling				Without Rescheduling			
No.of Pres.	Objec. Value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	38	0	0	No	38	0	0	No
3	38	0	0	No	38	0	0	No
6	23	0	6	No				Yes
9	12	0	29	No				
12	0	0	65	No				
15				Yes				

	With QoS functions				With a Unity QoS			
No.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	38	0	0	No	0	0	0	No
3	38	0	0	No	0	0	0	No
6	23	0	6	No	4	0	6	No
9	12	0	29	No	9	3	29	No
12	0	0	65	No	1	2	77	No
15				Yes				Yes

10. Large values and small S.D.of intervals,vary large values and small S.D.of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	20000	20000	Text
2	3	4	900000	150000	Audio
3	5	6	45000	45000	Text
4	7	8	22500	22500	Text
5	9	10	337500	56200	Audio
6	11	12	506100	84300	Audio
7	13	14	600000	100000	Video
8	15	16	30000	30000	Text
9	17	18	450000	75000	Audio
10	19	20	675000	112500	Video

QoS function for the constraint between N1 and N2: x - 998 and 1002 - x

QoS function for the constraint between N3 and N4: 3x – 3006 and 3018 - 3x

QoS function for the constraint between N5 and N6: x - 1002 and 1006- x

QoS function for the constraint between N7 and N8: x - 1002 and 1006 - x

QoS function for the constraint between N9 and N10: 3x - 3018 and 3030 - 3x

QoS function for the constraint between N11 and N12: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N13 and N14: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N15 and N16: x - 998 and 1002 - x

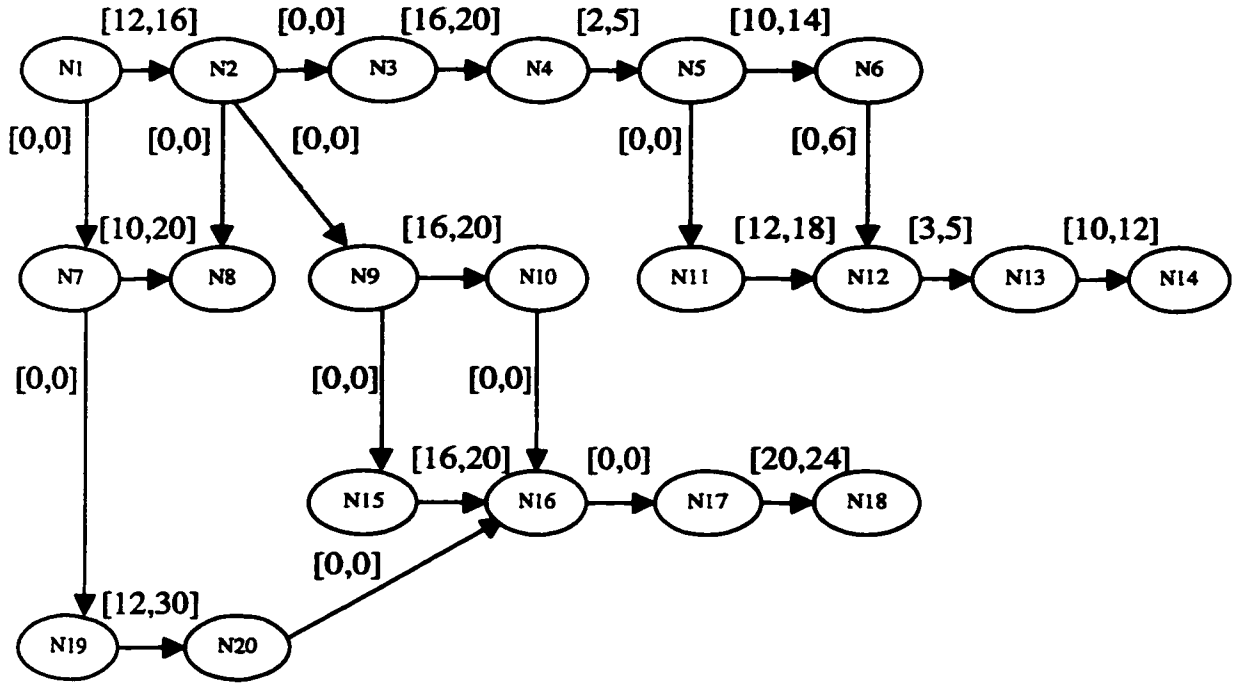
QoS function for the constraint between N17 and N18: 3x - 2994 and 3006 - 3x

QoS function for the constraint between N19 and N20: 3x - 3138 and 3138 - 3x

	With Rescheduling				Without Rescheduling			
No.of Pres.	objec. value	incons. Value	no.of resch.	Abort	objec. value	Incons. Value	no.of resch.	Abort
1	38	0	0	No	38	0	0	No
3	38	0	0	No	38	0	0	No
6	23	0	6	No				Yes
9				Yes				
12								
15								

	With QoS functions				With a Unity QoS			
No.of Pres.	objec. value	incons. Value	no.of resch.	Abort	objec. value	Incons. Value	no.of resch.	Abort
1	38	0	0	No	0	0	0	No
3	38	0	0	No	0	0	0	No
6	23	0	6	No	4	0	6	No
9				Yes	9	3	29	No
12								Yes

11.Small values and small S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	100	100	Text
2	3	4	9000	9000	Video
3	5	6	10000	2000	Video
4	7	8	6000	1000	Audio
5	9	10	300	300	Text
6	11	12	4000	100	Audio
7	13	14	600	600	Text
8	15	16	6000	500	Audio
9	17	18	100	100	Text
10	19	20	30000	1000	Video

QoS function for the constraint between N1 and N2: x - 12 and 16 - x

QoS function for the constraint between N3 and N4: x - 16 and 20 - x

QoS function for the constraint between N5 and N6: 3x - 30 and 42 - 3x

QoS function for the constraint between N7 and N8: 3x - 30 and 60 - 3x

QoS function for the constraint between N9 and N10: x - 16 and 20 - x

QoS function for the constraint between N11 and N12: 3x - 36 and 54 - 3x

QoS function for the constraint between N13 and N14: x - 10 and 12 - x

QoS function for the constraint between N15 and N16: 3x - 48 and 60 - 3x

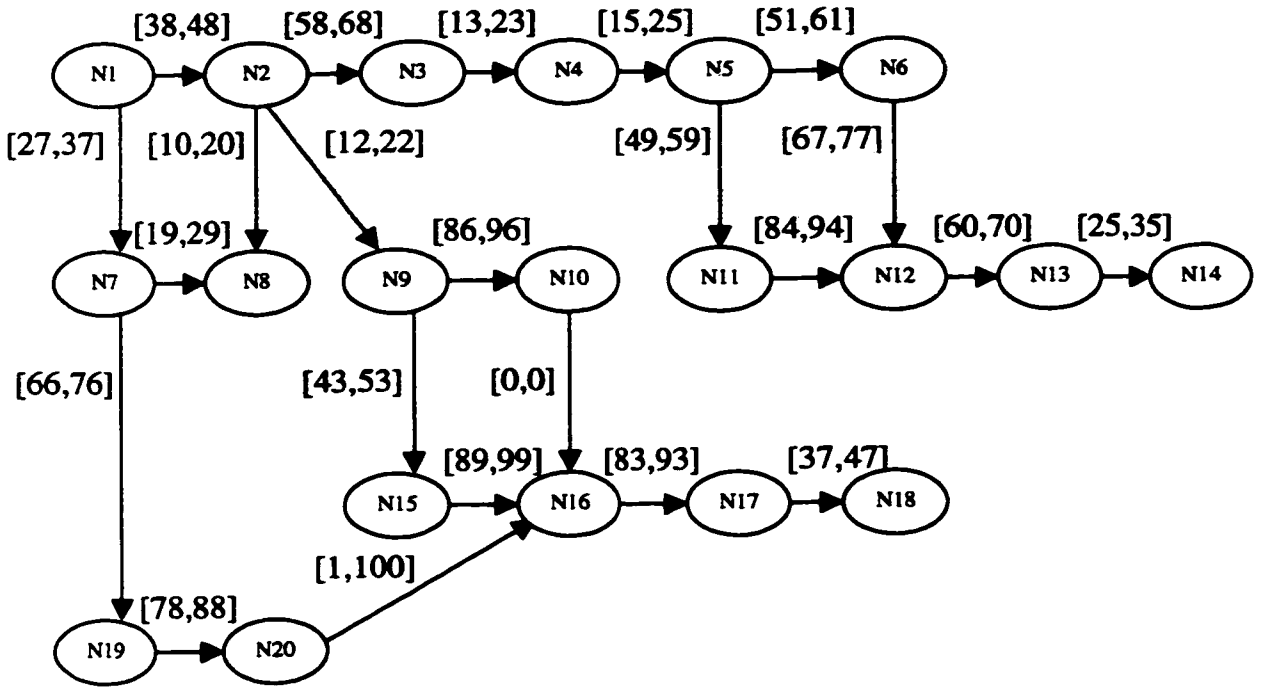
QoS function for the constraint between N17 and N18: x - 20 and 24 - x

QoS function for the constraint between N19 and N20: 3x - 36 and 90 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	incons. Value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	36	0	0	No	36	0	0	No
3	36	0	0	No	36	0	0	No
6	32	0	3	No				Yes
9	13	0	13	No				
12				Yes				
15								

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	36	0	0	No	12	0	0	No
3	36	0	0	No	12	0	0	No
6	32	0	3	No	10	0	3	No
9	13	0	13	No				Yes
12				Yes				
15								

12.Small values and large S.D. of intervals, small values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	100	100	Text
2	3	4	9000	9000	Video
3	5	6	10000	2000	Video
4	7	8	6000	1000	Audio
5	9	10	300	300	Text
6	11	12	4000	100	Audio
7	13	14	600	600	Text
8	15	16	6000	500	Audio
9	17	18	100	100	Text
10	19	20	30000	1000	Video

QoS function for the constraint between N1 and N2: x - 38 and 48 - x

QoS function for the constraint between N3 and N4: x - 13 and 23 - x

QoS function for the constraint between N5 and N6: 3x - 153 and 183 - 3x

QoS function for the constraint between N7 and N8: 3x - 57 and 87 - 3x

QoS function for the constraint between N9 and N10: x - 86 and 96 - x

QoS function for the constraint between N11 and N12: 3x - 252 and 282 - 3x

QoS function for the constraint between N13 and N14: x - 25 and 35 - x

QoS function for the constraint between N15 and N16: 3x - 267 and 297 - 3x

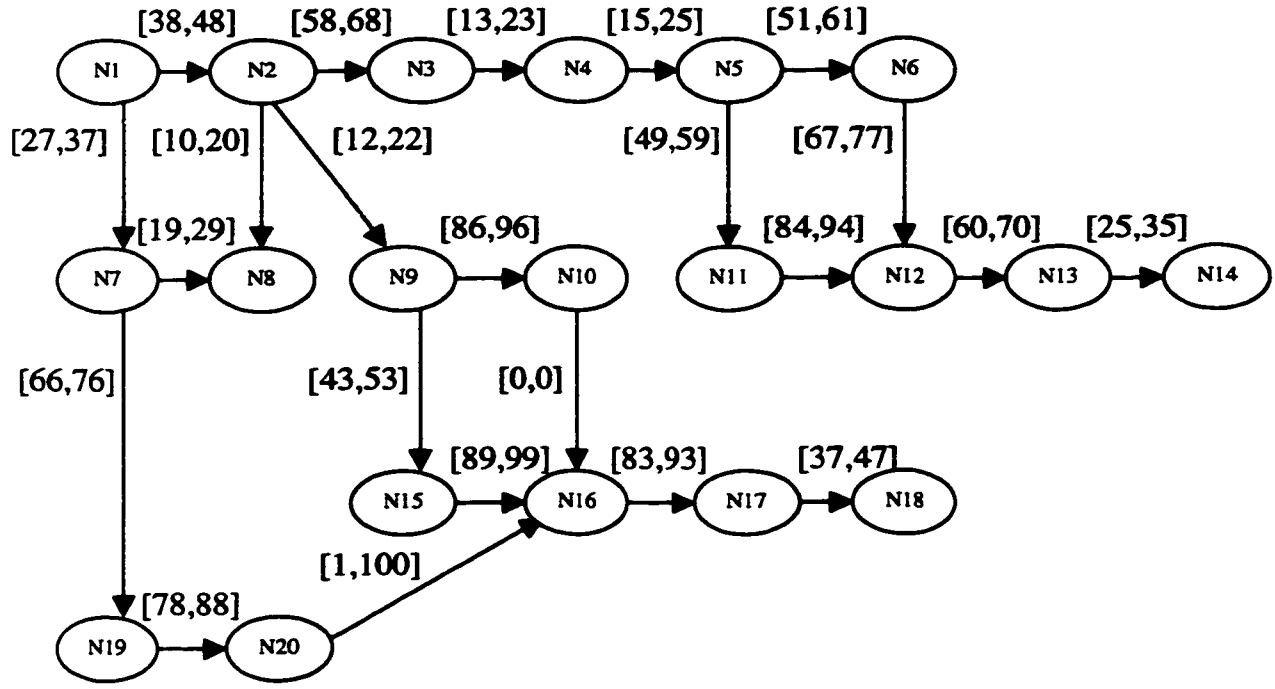
QoS function for the constraint between N17 and N18: x - 37 and 47 - x

QoS function for the constraint between N19 and N20: 3x - 234 and 264 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	85	0	0	No	85	0	0	No
3	85	0	0	No	85	0	0	No
6	85	0	0	No	85	0	0	No
9	79	0	4	No				Yes
12	54	0	29	No				
15	45	0	59	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	85	0	0	No	6	0	0	No
3	85	0	0	No	6	0	0	No
6	85	0	0	No	6	0	0	No
9	79	0	4	No	12	0	4	No
12	54	0	29	No	18	0	32	No
15	45	0	59	No	15	0	59	No

13. Small values and large S.D. of intervals, small values and small S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	100	100	Text
2	3	4	900	900	Video
3	5	6	1000	200	Video
4	7	8	600	100	Audio
5	9	10	300	300	Text
6	11	12	400	100	Audio
7	13	14	600	600	Text
8	15	16	600	500	Audio
9	17	18	100	100	Text
10	19	20	3000	1000	Video

QoS function for the constraint between N1 and N2: x - 38 and 48 - x

QoS function for the constraint between N3 and N4: x - 13 and 23 - x

QoS function for the constraint between N5 and N6: 3x - 153 and 183 - 3x

QoS function for the constraint between N7 and N8: 3x - 57 and 87 - 3x

QoS function for the constraint between N9 and N10: x - 86 and 96 - x

QoS function for the constraint between N11 and N12: 3x - 252 and 282 - 3x

QoS function for the constraint between N13 and N14: x - 25 and 35 - x

QoS function for the constraint between N15 and N16: 3x - 267 and 297 - 3x

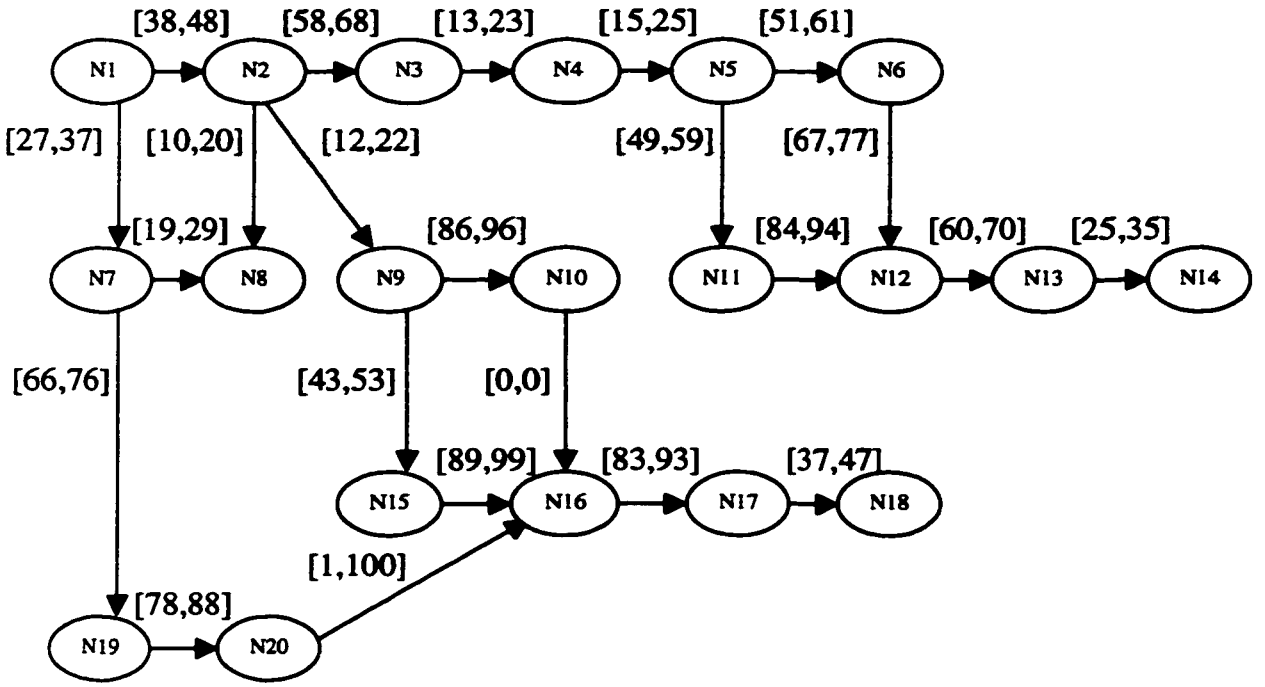
QoS function for the constraint between N17 and N18: x - 37 and 47 - x

QoS function for the constraint between N19 and N20: 3x - 234 and 264 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	85	0	0	No	85	0	0	No
3	85	0	0	No	85	0	0	No
6	84	0	1	No	85	0	0	No
9	83	0	2	No				Yes
12	79	0	4	No				
15	73	0	8	No				

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. value	incons. value	no.of resch.	Abort
1	85	0	0	No	6	0	0	No
3	85	0	0	No	6	0	0	No
6	84	0	1	No	7	0	1	No
9	83	0	2	No	8	0	2	No
12	79	0	4	No	9	0	4	No
15	73	0	8	No	12	0	8	No

14. Small values and large S.D. of intervals, large values and large S.D. of object size



No.of obj.	Start from	End to	Size of obj.	Size of first chunk	Type of obj.
1	1	2	10000	10000	Text
2	3	4	90000	90000	Video
3	5	6	100000	20000	Video
4	7	8	60000	10000	Audio
5	9	10	30000	30000	Text
6	11	12	40000	10000	Audio
7	13	14	60000	60000	Text
8	15	16	60000	50000	Audio
9	17	18	10000	10000	Text
10	19	20	30000	10000	Video

QoS function for the constraint between N1 and N2: x - 38 and 48 - x

QoS function for the constraint between N3 and N4: x - 13 and 23 - x

QoS function for the constraint between N5 and N6: 3x - 153 and 183 - 3x

QoS function for the constraint between N7 and N8: 3x - 57 and 87 - 3x

QoS function for the constraint between N9 and N10: x - 86 and 96 - x

QoS function for the constraint between N11 and N12: 3x - 252 and 282 - 3x

QoS function for the constraint between N13 and N14: x - 25 and 35 - x

QoS function for the constraint between N15 and N16: 3x - 267 and 297 - 3x

QoS function for the constraint between N17 and N18: x - 37 and 47 - x

QoS function for the constraint between N19 and N20: 3x - 234 and 264 - 3x

	With Rescheduling				Without Rescheduling			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. Value	incons. value	no.of resch.	Abort
1	85	0	0	No	85	0	0	No
3	85	0	0	No	85	0	0	No
6				Yes				Yes
9								
12								
15								

	With QoS functions				With a Unity QoS			
no.of Pres.	objec. value	incons. value	no.of resch.	Abort	objec. Value	incons. value	no.of resch.	Abort
1	85	0	0	No	6	0	0	No
3	85	0	0	No	6	0	0	No
6				Yes	0	11	45	No
9								Yes
12								
15								