Adaptability Management and Deterministic Scheduling of Media Flows on Parallel Storage Servers

Costas Mourlas, Department of Computer Science, University of Cyprus, 75 Kallipoleos str., CY-1678 Nicosia, Cyprus Tel. ++(357)-2-892228, Fax. ++(357)-2-339062 E-mail: mourlas@ucy.ac.cy

Abstract: We study a new design strategy for the implementation of Parallel Media Servers with a predictable behavior. This strategy makes the timing properties and the quality of presentation of a set of media streams predictable. The proposed strategy provides deterministic guarantees and service reliability for each stream that can't be compromised by server contention. Our real-time scheduling approach exploits the performance of parallel environments and seems a promising method that brings the advantages of parallel computation in media servers. The proposed mechanism provides deterministic service for both Constant Bit Rate (CBR) and Variable Bit Rate (VBR) streams. We present an efficient placement strategy for data frames as well as an adaptability strategy that allows appropriate frames to be dropped without sacrificing the ability to present multimedia applications predictably in time. A prototype implementation of the proposed parallel media server illustrates the concepts of server allocation and scheduling of continuous media streams.

Διαχείριση Προσαρμοστικότητας και Ντετερμινιστικής Χρονοδρομολόγησης Ροών Συνεχών Μέσων σε Παράλληλους Εξυπηρετητές Αποθήκευσης

Κώστας Μουρλάς, Πανεπιστήμιο Κύπρου, Τμήμα Πληροφορικής Καλλιπόλεως 75 ,Τ.Θ. 20537, CY-1678 Λευκωσία, Κύπρος Τηλ. ++(357)-2-892228, Fax. ++(357)-2-339062 E-mail: `mourlas@ucy.ac.cy`

Περίληψη: Μελετούμε μία νές στρατηγική σχεδιασμού για την υλοποίηση Παράλληλων Εξυπηρετών Πολυμέσων οι οποίοι θα έχουν προβλέψιμη συμπεριφορά. Η προτεινόμενη στρατηγική θα επιτρέπει την προβλεψιμότητα της παρουσίασης εφαρμογών πολυμέσων όσον αφορά τις χρονικές ιδιότητες και εξασφάλιση σταθερής ποιότητας. Οι εξυπηρετητές των συνεχών μέσων θα μπορούν να εξασφαλίζουν την χρονική ορθότητα των συνεχών μέσων (continuous media) που εξυπηρετούν, ανεξάρτητα του φόρτου εργασίας τους. Η προτεινόμενη στρατηγική βασίζεται σε ένα αλγόριθμο χρονοδρομολόγησης πραγματικού χρόνου ο οποίος εκμεταλεύεται τις δυνατότητες ενός παράλληλου περιβάλλοντος εκτέλεσης. Η προτεινόμενη μέθοδος παρέχει ντετερμινιστικές υπηρεσίες τόσο για ροές (streams) σταθερού ρυθμού bit (CBR) όσο και για ροές μεταβλητού ρυθμού bit (VBR). Παρουσιάζεται μία αποδοτική στρατηγική τοποθέτησης των δεδομένων των μέσων (media) στούς δίσκους ενός παράλληλου εξυπηρετητή καθώς επίσης και ο μηχανισμός προσαρμογής στην ανάκτηση των δεδομένων δυναμικά και ανάλογα με το φόρτο και τη χρήση του εξυπηρετητή. Με αυτό το τρόπο δεν καταστρέφεται η χρονική ορθότητα και η ποιότητα παρουσίασης εφαρμογών πολυμέσων. Μία ενδεικτική υλοποίηση παρουσιάζει την ορθότητα και την ευελιξία της προτεινόμενης στρατηγικής.

Adaptability Management and Deterministic Scheduling of Media Flows on Parallel Storage Servers

Costas Mourlas
Department of Computer Science, University of Cyprus,
75 Kallipoleos str., CY-1678 Nicosia, Cyprus
mourlas@ucy.ac.cy

Abstract

We study a new design strategy for the implementation of Parallel Media Servers with a predictable behavior. This strategy makes the timing properties and the quality of presentation of a set of media streams predictable. The proposed strategy provides deterministic guarantees and service reliability for each stream that can't be compromised by server contention. Our real-time scheduling approach exploits the performance of parallel environments and seems a promising method that brings the advantages of parallel computation in media servers. The proposed mechanism provides deterministic service for both Constant Bit Rate (CBR) and Variable Bit Rate (VBR) streams. We present an efficient placement strategy for data frames as well as an adaptability strategy that allows appropriate frames to be dropped without sacrificing the ability to present multimedia applications predictably in time. A prototype implementation of the proposed parallel media server illustrates the concepts of server allocation and scheduling of continuous media streams.

1 Introduction

Continuous media servers differ enough from traditional storage servers since they store and manipulate continuous media data (video and audio) which consist of *streams* of media quanta (video frames and audio samples) that must be presented using the same timing sequence with which they were captured. For example, throughput must be maintained continuously for acceptable video playouts, and jitter must be kept within strict bounds of the order of 10ms for digital audio to be intelligible to humans. This implies that special scheduling and resource allocation strategies must be provided by the CM server such that the required CM data will be available for the time they are needed.

Hence, media servers need to ensure that the retrieval and storage of such CM streams proceed at their pre-specified real-time rates.

Our current work is focused on the design and implementation of a predictable parallel media server. We focus mainly on resource management of the parallel server in order to provide on-demand support for a large number of concurrent continuous media objects in a *predictable* manner. With the ability to manage parallel data retrievals on media servers that satisfy the real-time requirements of each stream we could be able to concurrently support more predictable continuous media applications than on traditional single processing servers.

In a subsequent step, we extend our resource management strategy to provide adaptability. Instead of rejecting requests, adaptability allows more requests to be served by a suitable choice of frame dropping. The proposed adaptability management provides this feature without sacrificing the ability to present multimedia applications predictably in time. Our resource and adaptability management strategies have been especially designed for parallel media servers and support both CBR and VBR encoded media streams (video and audio) in a predictable manner.

2 The Architecture of the Parallel Media Server

One common server architecture is the single processing model. However, this single processor server model has its limitations like performance, scalability, low transfer rate and low capacity. Recently, much research has been made on the topic of parallel systems in the community of parallel computing. In order to design a general purpose architecture which can be adapted to the current user requirements, a scalable parallel multimedia server shall be designed.

We will use the traditional model for a parallel media server previously described in [13, 5]. In that architecture there exist three kinds of nodes: storage nodes, delivery nodes and one control node (see Figure 1). The three kinds of nodes are explained in greater detail below:

Storage nodes are responsible for storing video and audio clips, retrieving requested data blocks and sending them to delivery nodes within a time limit. In addition, partitioned media blocks are wide striped among storage nodes in a round-robin fashion to balance the workload.

Delivery nodes are responsible for serving stream requests that have been previously accepted for service. Their main function is to request the striped data from the storage nodes through the internal interconnection network, re-sequence the packets received if necessary and then send the packets over the wide area network to the clients.

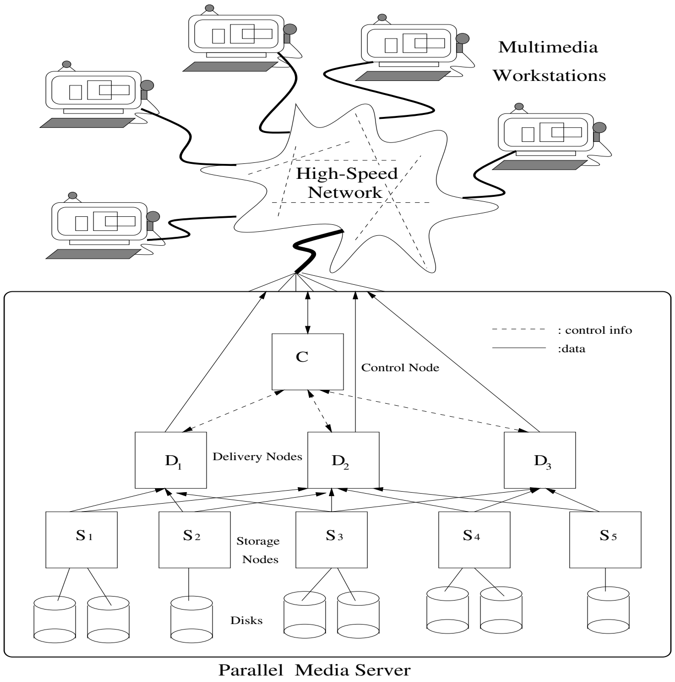

Figure 1: *The logical model of the parallel media server*

The Control node receives all incoming requests for media objects. It has knowledge of which storage node stores the first data block of the object and the workload of the delivery and storage nodes. In a typical request-response scenario, the control node receives a request for a media object. If the resource requirements of the request are consistent with the system load at that time, then the request is accepted. A delivery node to serve the stream is chosen by the control node and the delivery node then takes over the authority of serving the stream. To that end, it retrieves the stream fragments from the storage nodes and transmits them at the required rate to the client.

The logical storage and delivery nodes can be mapped to different as well as to the same physical node. The model where a node can be both a storage node and a delivery node is called "flat" architecture and it is more suitable to be implemented on a cluster of workstations interconnected by high-speed links. In this paper, we are focused on "flat" architectures.

Due to the fact that many parallel tasks share the server resources (storage and delivery nodes) and execute periodic reads for data retrieval to satisfy the stream's real-time constraints, it happens for one task to wait till some resources of the media server become available by other tasks. Since tasks are inter-dependent and share both storage and delivery nodes our main problem is the scheduling of the incoming requests to improve performance and high level of system utilization. The required real-time scheduling algorithm needs to prevent under-utilization of the resources and ensure load balancing. These problems are addressed in the following section.

3 The Proposed Scheduling Algorithm

It is well understood that scheduling policies are critical to performance of continuous media servers. Without scheduling and resource management, streams may conflict each other. Therefore may be delayed and the quality of service cannot be guaranteed. Furthermore, buffers are used for a smooth delivery of data to the clients through the storage and delivery nodes. Continuous media streams that are not well scheduled may require large buffer space. The work we present here on scheduling of a parallel media server is focused on deterministic guarantees, so that the application can maintain the requested QoS level without encountering unpredictable delay and jitter while reproducing the video display and audio sound. The proposed scheduling algorithm guarantees the QoS of every (accepted) stream, efficiently utilizes server resources, reduces the required buffer size and increases system throughput.

As mentioned earlier, the data is compressed and striped across all storage nodes in a round-robin fashion. Although data blocks are wide striped, without properly scheduling of data retrievals, resource conflicts may be occurred such as *port contention* where two storage nodes are transmitting to a single delivery node at the same time. Another resource conflict that may also happen is *disk contention* where more than one request retrieve blocks from the same storage node at the same time instance.

The work described in this paper, is concentrated on the special case of conflict-free scheduling that provides deterministic guarantees for both CBR and VBR stream requests. It is well understood that providing deterministic service for CBR streams is easier due to the fact that a CBR stream requests the same amount of data in every interval. For presentational purposes, the initial version of our scheduling algorithm that schedules CBR requests is presented first. In a following subsection we extend our scheduling algorithm to include VBR streams taking into account the fluctuations in the bit rates of multiple requests that may overload throughput capacity of the storage nodes.

3.1 Deterministic Guarantees for CBR Streams Encoded at Different Playback Rates

We will describe how requests for media streams can be modeled as a set of periodic tasks and we give a formal evaluation of some components such as the period and the data retrieval section of each task. Time is divided into time frames (or rounds) where the length of every time frame T_i equals to T which is a constant value. R_i is the required playback rate for stream s_i that has been pre-determined during the compression phase of that stream. Note that, for a CBR stream s_i, the value R_i is constant during the length of the stream. Different CBR streams stored in a media server usually have been encoded in different playback rates for different qualities of audio and video objects. Due to the fact that the data transfer rate of a single disk or a disk array can be much higher than the playback rate of a stream, multiple media streams can be served by a storage server in every T time units while the individual playback rate R_i is still preserved. Our aim in the design of a parallel media server is to supply the stream with enough data to ensure that the playback processes do not starve.

Therefore, every stream s_i is represented by a periodic task τ_i where in every period (i.e. in every time frame) T needs to retrieve $F_i = T * R_i$ amount of data to guarantee that the stream s_i will meet its real-time requirements. The above equation determines the stripe fragment size F_i of stream s_i which is different in general for every stream according to its playback rate R_i. Every media stream s_i is striped across all nodes in a round-robin fashion where the stripe fragment size of s_i equals to F_i. The average time to retrieve F_i bytes from the storage node and transmit them to a delivery node is given by equation

$$t_s^i = t_{avg\_seek} + t_{avg\_rot} + t_{r\_F_i} + t_{nw\_Fi} \tag{1}$$

where t_{avg_seek} and t_{avg_rot} are the average seek and rotational latencies for the disks being used, $t_{r\_F_i}$ is the disk data transfer time for F_i bytes and t_{nw_Fi} is the internal network latency to transport F_i bytes from a storage node to a delivery node. Thus, t_s^i is the length of the data retrieval section of the periodic task τ_i. Note that the equation 1 uses average seek and rotational latencies for disk accesses. Since these latencies are variable, there will be boundary conditions when the time to retrieve F_i bytes is much more (less) than the average value. If some clients require strict performance guarantees, then one can categorize users into those requiring hard and soft deadlines and use the maximum values of the disk overheads for admitting such users.

Since the stripe fragments of a continuous media are consecutively distributed in all N storage nodes, if a task τ_i at time frame m retrieves data from node k, it will retrieve data from node $(k + l)$ mod N at time frame $(m + l)$. A complete schedule is represented by a schedule table consisting of N consecutive time frames. Let u_i be the set of tasks allocated to the delivery node i for service. We define as the utilization factor U_i of a delivery node i, the sum given by the formula:

$$U_i = \sum_{\tau_j \in u_i} \frac{t_s^j}{T}, \qquad 0 \le i \le N-1 \tag{2}$$

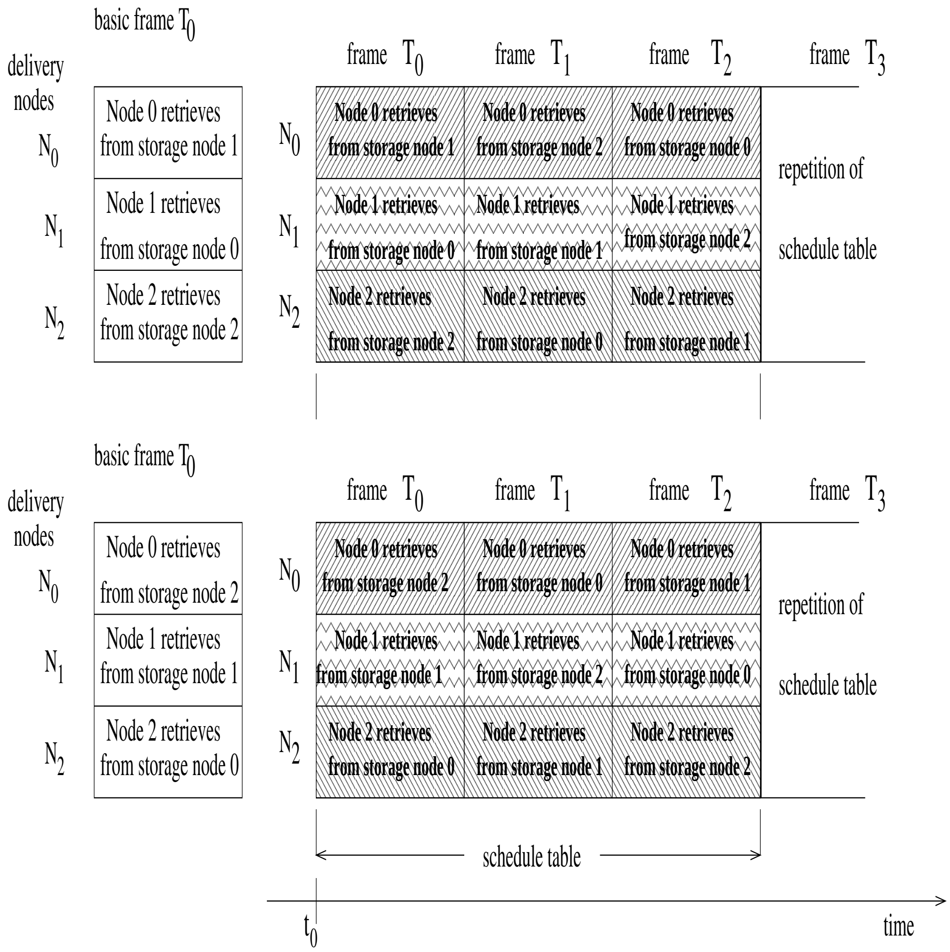

Figure 2: *Equivalent schedule tables starting from a different basic time frame T_0.*

The value U_i of a delivery node i changes only when a new request is allocated to the delivery node i by the control node, or when an existing request completed its execution and quits. U_i represents the load of the delivery node i and its value can never be greater than one. Note that, the utilization factor of a storage node varies from one time frame to the other. More precisely, the load of one storage node in the frame T_i moves to the next storage node in frame T_{i+1} and returns in frame T_{i+N}.

Our current work is concentrated on the special case of scheduling called conflict-free scheduling [13]. It is an extension of the work presented in [8] and [10] based on a conflict-free stream scheduling algorithm that eliminates contentions so that high system performance and stream throughput can be achieved. The algorithm guarantees that once the first round has a conflict-free scheduling, the following time frames will not have conflict. Our first extension of that scheduling scheme is described in the following paragraphs and provides better flexibility and better performance for large-scale parallel media servers. Based on the extended scheduling scheme we will be able to provide streams of different playback rates and make maximum utilization of resources which are not possible in the original version of the algorithm described in [8] and [10]. In the next subsection, we extend further

Stream Parameters			
Request	Type	Playback Rate	Starting Storage Node
r_0	video	1.5 Mbits/sec	2
r_1	audio	0.6 Mbits/sec	1
r_2	audio	0.4 Mbits/sec	0
r_3	video	1.2 Mbits/sec	0
r_4	video	2.2 Mbits/sec	1

Table 1: *The Stream Parameters used in the Example*

our scheduling strategy to accommodate VBR stream requests.

A conflict-free schedule is a schedule that in every time instance the following scenario will never occur: two media streams request data from the same storage server or two storage servers transmit data to the same delivery node. In order to construct such a schedule we implement every frame (or round) in such a way that only one storage node transmits data to one delivery node. Note that, a starting sequence which designates the transmission order between storage nodes and delivery nodes needs to be assigned at the first basic time frame T_0. Different but equivalent basic time frames exist each one with a different starting sequence and any of that frames can be selected as the basic frame T_0. Since the blocks of the media streams are consecutively distributed in all N storage nodes, when delivery node 0 schedules a request that retrieves a block from storage node 1 the same request retrieves blocks from storage nodes 2,3,...,N-1,0 in the next N-1 frames (see Figure 2).

The proposed algorithm schedules the stream requests as follows: When a new request r_k for a media object arrives where its starting block is stored in node j, the first step is to test schedulability of the new request. The control node checks the loads of the delivery nodes and finds the node i, $(0 \leq i \leq N-1)$ with the minimum load U_i. Then, it checks if the condition $U_i + \frac{t_s^k}{T} \leq 1$ is satisfied for that node. If the condition is satisfied then the node i is declared as the delivery node of the stream s_k and it will serve together with the previous streams the new one during its lifetime. The new request r_k starts receiving data when delivery node i is connected with storage node j for first time after the receipt of request r_k and the schedule table is updated accordingly. Notice the possibility to delay the beginning of service for a request till delivery node is connected for the first time after the receipt of the request with the corresponding storage node. In case that the above condition cannot be satisfied the request is rejected or postponed for later service. An important property of the proposed algorithm is that when the control node finds a delivery node i to serve the request r_k and the condition $U_i + \frac{t_s^k}{T} \leq 1$ can be satisfied, immediately it is guaranteed that the new load $U_i' = U_i + \frac{t_s^k}{T}$ can be accommodated also by the storage nodes.

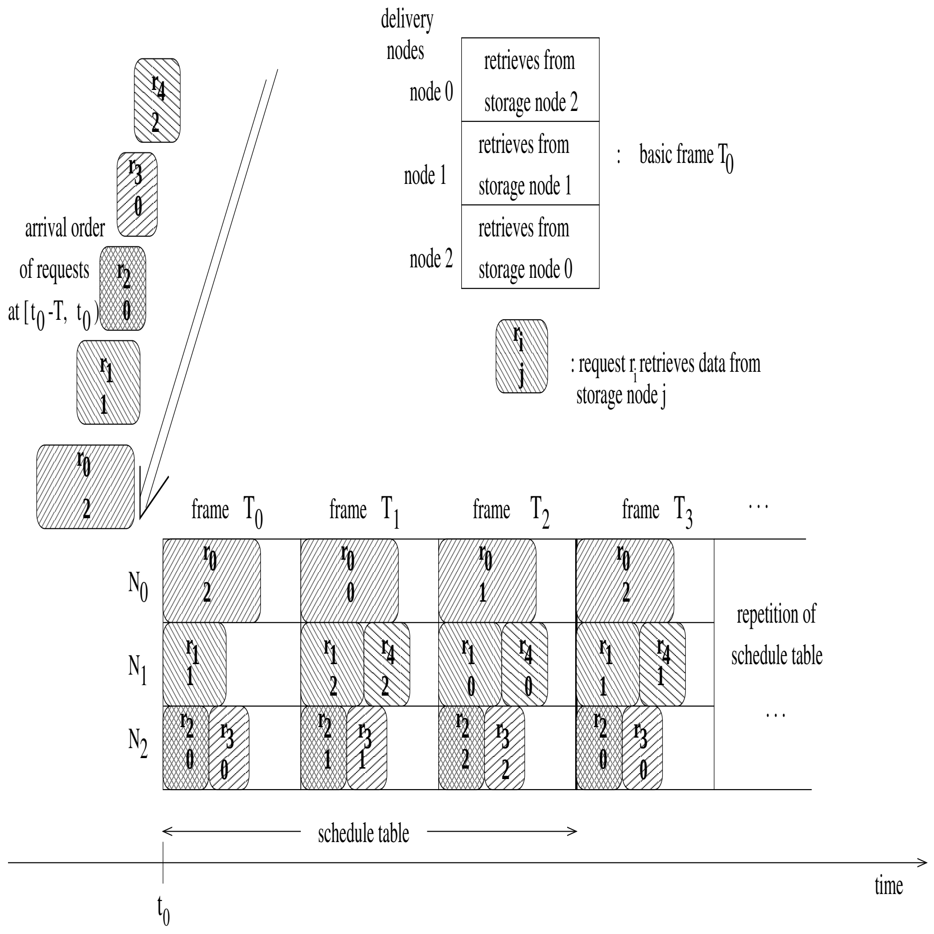

Figure 3: *The basic frame T_0, the arrival order of the requests and the complete schedule of the example*

The proposed conflict-free scheduling algorithm can be illustrated by a single example. Suppose that a parallel media server with a "flat" architecture supports 3 nodes (N=3). The stream parameters of the example are presented in Table 1. Suppose that an empty basic frame T_0 is given and all the media requests arrive in sequence during the time interval $[t_0 - T, t_0)$. An entry in a time frame (i.e. a shaded area) shows the data retrieval section of the request and the storage node number from where the stripe fragment is retrieved (see Figure 3). The retrieval of the stripe fragments of a single stream are separated by one time frame. In our example, delivery node 0 schedules the first request r_0 that retrieves a stripe fragment from storage node 2 in time frame T_0. The same request retrieves blocks from storage nodes 0 and 1 in the next two frames. A complete schedule is represented by a schedule table consisting of 3 time frames (see Figure 3). Notice also that request r_4 is delayed for T time units before it is served.

3.2 Deterministic Guarantees for VBR Streams

The problem of providing deterministic guarantees for VBR streams is harder due to the following two reasons:

1. the load of a stream on the storage units varies from one round to the other, and
2. scheduling the first block of a VBR stream does not mean that the rest blocks of the stream can be scheduled.

One approach for the solution of the problem is to compute the peak rate of the stream and reserve enough bandwidth on storage nodes to satisfy the peak requirements of the stream. This pessimistic approach results to the underutilization of resources since the peak demand is observed only for short durations compared to the whole duration of the stream. When many streams are served on the parallel server, it is very possible that the peak demands of the streams do not overlap with each other. Thus, it is true that we can actually serve more streams than it is allowed by the peak-rate allocation, without reducing the quality provided by the server. We propose an extension of the previous scheduling algorithm for CBR streams that allows the system to increase the number of accepted requests for streams while providing deterministic service.

In the previous subsection, every CBR stream s_i is represented by a periodic task τ_i where in every time frame T needs to retrieve a constant data length block F_i determined by the equation $F_i = T * R_i$. R_i is the required playback rate for stream s_i which is a constant value for every different CBR stream. Using VBR streams, the bit rate R_i is variable which means that the stripe fragment size for VBR streams is also variable and thus the workload for the storage devices changes from one frame to the other.

Our approach considers the variations in loads and provides guarantees for VBR streams as follows: Time is divided as described above into time frames of equal size T. We introduce the parameter FR_i which denotes the frame rate of real-time playback (frames per second) for a video stream s_i (or samples per second for audio stream) determined when the media was captured. Thus, the number of frames included in every stripe fragment of the media stream s_i is given by the number $FR_i * T$. Our new requirement that we set here is that the result of the product $FR_i * T$ must always be an integer value for all the streams stored in our parallel server. The stripe fragment size $F_i[k]$ on round k is given by the expression: $F_i[k] = sizeof(FR_i * T\ frames\ on\ round\ k)$. We therefore store and read the data in units of $F_i[k]$ which is of variable length on every different round. Every VBR stream s_i is striped across all nodes in a round-robin fashion where the stripe fragment size for round k equals to $F_i[k]$. Notice that just as with CBR streams, data required during a round is located on a single storage node. The time duration $t_s^i[k]$ to retrieve $F_i[k]$ bytes from the storage node and transmit them to delivery node during the k_{th} round is given by equation 1 described in the previous section.

In our approach, a presentation requiring service for a VBR stream supplies the media server with a *load vector* for that stream according to its demands. More precisely, the load vector $LV_i[k]$, $0 \leq k < dur(s_i)$ describes the time required for a storage node to retrieve and transmit $F_i[k]$ data units to a delivery node on each round k. The term $dur(s_i)$ defines the length of the stream s_i in rounds. The load vector $LV_i[]$ for stream s_i can be stored on the Control Node in a form of a special file. We can easily conclude that compared to the size of the video or audio file, the size of the load vector file is not significant.

The Control Node keeps track also of the utilization of every delivery node i of the server in the form of a utilization vector $U_i[]$. The utilization vector $U_i[]$ for delivery node i stores the actual utilization of that node in each round over sufficient period of time. Let u_i be the set of tasks allocated to the delivery node i for service. We define as the utilization $U_i[k]$ of a delivery node i during the round k, the sum given by the formula:

$$U_i[k] \;=\; \sum_{\tau_j \in u_i} \frac{t_s^j[k]}{T}, \qquad 0 \leq i \leq N-1 \tag{3}$$

The values of the $U_i[]$ vector of a delivery node i are modified when a new request is allocated to the delivery node i by the control node, or when an existing request completed its execution and quits. In addition, the utilization of a delivery node changes from one time frame to the other due to the variable bit rate of the streams. $U_i[k]$ represents the load of the delivery node i on round k and its value can never be greater than one. The utilization vector $U_i[]$ of a delivery node i stores the current as well as the future utilization values for that delivery node. Before a stream is accepted by the control node, its load vector is combined with the utilization vector of every delivery node to check if there exists a delivery node where its new load never exceeds its capacity (i.e. its utilization is always lower than or equal to one during the length of the candidate stream). Let $U_i[j]$ denote the utilization of delivery node i in round j when a new request for stream s_r arrived at the server. Let the starting block of the stream s_r be on a storage node where delivery node i will be connected to that node after k time frames ($0 \leq k \leq N-1$, where N is the total number of storage nodes). Then, the stream can be admitted if there exists a delivery node i that all the following m conditions:

$$U_i[j+k+m] \;+\; \frac{LV_r[m]}{T} \leq 1, \qquad 0 \leq m < dur(s_r) \qquad where \tag{4}$$

$$U_i[j+k+m] = \sum_{\tau_j \in u_i} \frac{t_s^j[j+k+m]}{T} \quad and \quad LV_r[m] = t_s^r[m]$$

can be satisfied. The term k denotes the startup latency for that stream. In case that multiple delivery nodes satisfy the above conditions, the delivery node with the minimum value of k can be selected to provide the minimum startup latency. If the request is accepted then the utilization vector of the selected delivery node is updated accordingly. Let $dur(s_r)$ be the length of the stream s_r measured in time frames. The control node requires at most $N * dur(s_r)$ additions to determine if a stream can be accepted or not for service, where N is the number of delivery nodes of the parallel server.

4 Adaptability Management

A request of a media stream for service can be accepted or not following the aforementioned admission control policy. When a request is rejected it means that at least one of the m conditions given by (4) cannot be satisfied. In other words, if we accept such a request then there will be a future time instance lying between the time of acceptance and the duration of that stream when a storage node will be overloaded. Due to the fact that media presentations have not to be considered as hard real-time applications, we need a media server that enables us to drop the frame rate of a candidate stream to smaller values rather than rejecting that request.

Thus, we need to extend our approach on deterministic scheduling of media streams in order to support adaptability of media flows without sacrificing the ability to present multimedia applications predictably in time. Suppose that a request to serve a specific stream includes quality of service (QoS) specifications which are expressed with temporal and spatial resolutions. The temporal resolution can be expressed by the number of frames per second or sample rate and the spatial resolution can by expressed by data size or number of bits per pixel. For example, in one simple digital video application, the user may choose 22 frames per second for its temporal resolution and a spatial resolution of 160 by 120 pixels wide with an 8-bit color resolution. The video quality requirements can be specified using the following attributes:

fps : The value of fps defines the temporal resolution of a video presentation by giving the number of frames per second. The value of this attribute can be any positive integer or a range of positive integers. For example giving `fps=14-18` as attribute to a video object, it means that the accepted values for this video presentation can be any rate between 14 and 18 frames per second.

spatial-res : The *spatial-res* definition of a video presentation specifies the spatial resolution in pixels required for displaying the video object. If an ordered list of resolutions is given (e.g. *spatial-res=[180×130, 120×70]*) then the video object will be presented with the highest possible spatial resolution according to the availability of system resources and can be altered at run time.

color-res : This attribute specifies the color resolution in bits required for displaying the video object. The value must be greater than 0. Typical values are 2, 8, 24 .... If an ordered list of integer values is given (e.g. *color-res=[8,2]*) then the video object will be presented with a color resolution that is equal with one of the values of the list. During object presentation the highest color resolution is tried to be used first and this has to be decided at run time according to the availability of system resources.

The audio quality requirements can be specified using the attributes:

sample-rate : The value of *sample-rate* defines in kHz the rate that the analog signal is sampled. If we need, for example, telephone quality the analog signal should be sampled 8000 times per second (i.e. *sample-rate = 8*).

sample-size : This attribute specifies the sample size in bits of each sample. If an ordered list of integer values is given (e.g. *sample-size=[16,8]*) then each sample will be represented with a number of bits equal with one of the values given. For telephone quality, each sample of the signal is coded with 8 bits whereas for CD quality it is coded with 16 bits. The highest value that can be used for every sample has to be decided at run time according to the availability of the resources.

The above attributes form a complete set for QoS definition of every distinct continuous media that participate in a multimedia presentation. Using the above list of attributes and supporting an adaptability management module in our parallel media server we will be able to satisfy more requests by gracefully degrading their bit rates, instead of rejecting them. In this paper, we consider only temporal adaptability achieved by frame dropping. Thus, when a video request with attribute `fps=6-8` asks for service from the the parallel server, then the runtime system of the server will try to provide the best value in the range and it will be also authorized to modify this value at run-time towards the upper (8 fps) or the lower bound (6 fps) value according to the availability of the resources. The recommended strategy follows:

As we have already described, the number of frames included in every stripe fragment of a media stream s_i is constant given by the number $FR_i * T$. We store and read the data in units of $F_i[k]$ which are in general of variable length for every different round. Notice that data required during a round is located on a single storage node. In our adaptability strategy all the frames that are used for the lowest-rate reconstruction are grouped at the beginning of the data unit. Assume that a VBR stream has been compressed with $FR = 8fps$ and a request for that stream arrived at the control node of the server with quality attribute `fps=6-8`. Assume also that the length of every time frame T_i equals to 1 sec. The original frame pattern of that stream is $F_1F_2F_3F_4F_5F_6F_7F_8$.

By grouping the frames for each rate starting from 1 fps to 8 fps, our storage pattern becomes $F_1F_5F_7F_3F_4F_6F_2F_8$. Using this ordering, the system can degrade the request from 1 fps to the maximum possible frame rate 8 fps. For frame rate 1 fps only the first frame F_1 is retrieved, for frame rate 2 fps the frames F_1F_5 are retrieved, for frame rate 3 fps the frames $F_1F_5F_7$are retrieved, for frame rate 4 fps the frames $F_1F_5F_7F_3$ and so on. Delivery nodes re-sequence the retrieved frames and then send them to the clients. Notice that the frames are evenly spaced throughout the round and causes less jitter than would be caused by dropping the last frames of the original ordering. As shown above, we group all of the frames of each rate together in each read unit. In addition, all of the frames that are used for the lowest-rate reconstruction are grouped at the beginning of the data unit. In our example, where `fps=6-8` the runtime system is authorized to retrieve six $(F_1F_5F_7F_3F_4F_6)$, seven $(F_1F_5F_7F_3F_4F_6F_2)$ or eight $(F_1F_5F_7F_3F_4F_6F_2F_8)$ frames per second according to the availability of the resources.

The load vector of a stream becomes now a two-dimensional array $LV_i[j][k], 1 \leq j \leq max\_frames_i$, $0 \leq k < dur(s_i)$. The values in the first dimension $LV_i[1][k]$ specify the time required in every round k for a storage node to retrieve and transmit only one frame from the set of frames included in the stripe fragment $F_i[k]$. The values in $LV_i[2][k]$ specify the time to retrieve and transmit 2 frames and so on, until $LV_i[max\_frames_i][k]$ that specify the time required to retrieve and transmit all the frames in every stripe fragment $F_i[k]$. The size of the load vector $LV_i[][]$ is n times larger than the size of the previous one-dimensional vector $LV_i[]$, where n is the length of the first dimension of the vector $LV_i[][]$. If a small range of different frame rates for different qualities is supported then the size of every different load vector $LV_i[][]$ remains in acceptable levels.

Without adaptability, the load of a candidate stream is combined with the utilization vector of every delivery node to check if there exists a node where its new load never exceeds its capacity. If there exists such a node the request is accepted, otherwise it is rejected. Using a frame placement strategy similar to the one described above and using the two-dimensional *load vector* $LV_i[][]$ for each stream s_i the admission control policy can be improved providing different levels of adaptability. Suppose that a request specifies the expected quality of service from the server by giving the lower and the upper bound for the expected frame rate (e.g. $fps = 18 - 22$). Let min_fps and max_fps to indicate the lower and the upper bound of the expected quality for a video presentation. The admission control mechanism is authorized to select any value in the range $[min\_fps, max\_fps]$ when it is trying to find a delivery node where its new load never exceeds its capacity. Formally, a new set of conditions have to be satisfied similar to the ones described in (4). Let $U_i[j]$ denote the utilization of delivery node i in round j when a new request for stream s_r arrived at the server. Let the starting block of the stream s_r be on a storage node where delivery node i will be connected to that node after k time frames ($0 \leq k \leq N-1$, where N is the total number of storage nodes). Let $LV_r[n][k]$ be the load vector of stream s_r which includes the time required in every round k for a storage node to retrieve and transmit n frames grouped at the beginning of the stripe fragment $F_r[k]$. Then, the stream can be admitted if there exists a delivery node i that the following conditions are satisfied:

$$\forall m \quad \exists n \quad : U_i[j+k+m] \quad + \quad \frac{LV_r[n][m]}{T} \leq 1, \qquad where \tag{5}$$
$$0 \leq m < dur(s_r) \ , \qquad min\_fps_r \leq n \leq max\_fps_r$$

More advanced admission control strategies can be easily implemented using the proposed data structures and the frame placement policy. Such strategies will give the possibility for a stream to increase dynamically its frame rate during its presentation, towards the lower or the upper bound according to the availability of resources. These more sophisticated adaptive mechanisms are required in interactive environments where the user can suspend the presentation of a continuous media stream with no prior notification.

5 Experimental Results

A first prototype of the proposed parallel server has been implemented using a network of three Unix workstations, running Solaris 8 Operating System. It is a flat, cluster architecture where the three nodes are connected by a 100 Mbps Ethernet LAN. Each node is equipped with 128 MB RAM and one local SCSI disk. CBR media streams are partitioned as media blocks with size according to the playback rate of each stream. We used in our experiments CBR streams encoded in different playback rates with values between 1.17 Mbps and 1.95 Mbps. The selected length of every time frame T_i equals to 0.5 sec. The calculated stripe fragment size F_i of every CBR stream s_i is between 75 and 125 KB. Here, the length of a media block represents half of the second of playback time. In addition, we used in our experiments three VBR MPEG-encoded video streams, each one contained 40,000 samples at a frame rate of 24 frames per second (about half an hour playing time). The stripe fragment of every VBR stream contains 12 frames and it is varied from 42 KB to 280 KB. The load vector $LV_i[]$ of every VBR stream contains 3334 entries where each entry describes the time required for a storage node to retrieve and transmit the corresponding stripe fragment on a specific round. The stripe fragments of both CBR and VBR streams are wide striped into the three processing nodes in a round-robin fashion.

The prototype has been encoded in C using MPI calls that implement the communication between the three processing nodes. The pre-specified schedule table is sent once to the storage nodes and it is updated locally at the end of each round in order to save network bandwidth. The experiments test the performance of the parallel media server, evaluate the communication overhead and give us the opportunity to compare the theoretical results with the practical ones. We initially tested the performance of a single node media server using media streams with the previously described parameters. The maximum value of the throughput observed was 26 concurrent streams. In a subsequent step we tested the performance of the three-node cluster architecture where the three nodes were connected by a 100 Mbps Ethernet LAN. We evaluated the internal network latency to transport media blocks of variable length and the estimated latency values were between 10 ms and 15 ms. The computation overhead for the evaluation of conditions in (4) that determine the acceptance or not of a specific media request was not significant in our experiments (less than 1 ms). The overall throughput of the parallel server had a minimum value of 42 streams and a maximum value of 46 streams for the same set of requests. That variation of throughput is due to the fact that the internal network latency was not constant in our computing environment. Using an Ethernet Switch instead of an Ethernet LAN will result to a constant network latency value and absence of variation on stream throughput is expected for the parallel server. We have investigated the impact of the internal network latency on the stream throughput and the experiments leads us to the conclusion that internal network latency is the most critical factor for the performance of the server. Reducing the network latency improves the performance of the server reaching higher values of stream throughput. We have to notice here that the adaptability management module has not been implemented yet and it is expected to give even better values of stream throughput without sacrificing the ability to present the media streams predictably in time.

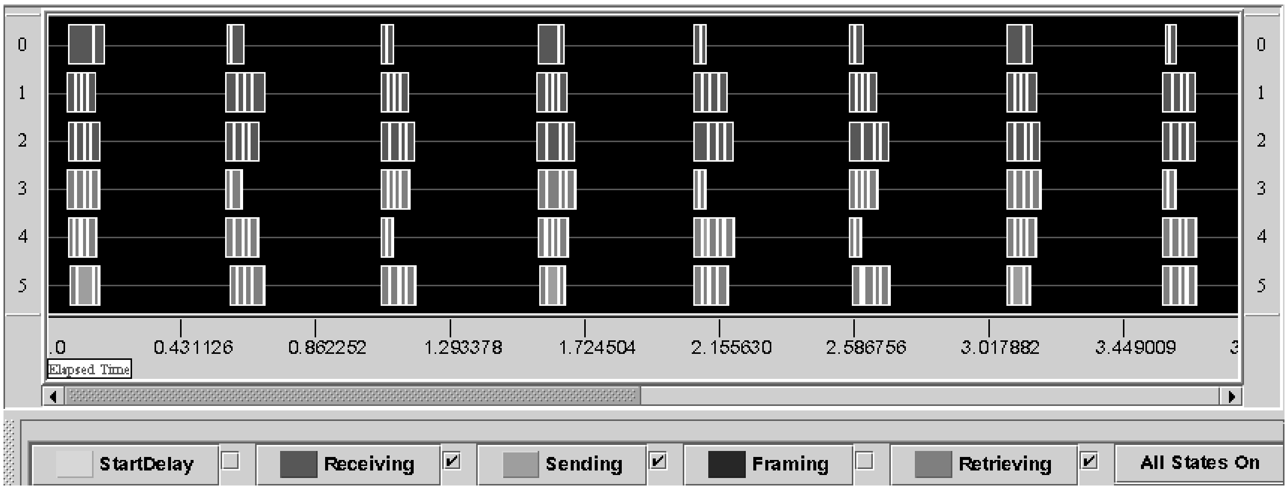

Figure 4: *The first 3.5 sec of the server execution serving 10 media streams*

Figure 4 is a graphical representation of the parallel server execution serving the first ten media requests. Everyone of the three nodes of the server executes two concurrent processes, the *Sender* process which is responsible for retrieving requested media blocks and sending them to the delivery node and the *Receiver* process which receives the media blocks from the *Sender* through the internal interconnection network.

In our future experiments we will implement the adaptability management module and we will check its effect on the stream throughput of the media server. We will also test the load balancing and the utilization of resources mainly at the point where the server will start rejecting additional requests. Other scheduling algorithms will be also implemented and their result will be compared with the proposed implementation.

6 Related Work

Much interesting work has been done in this field for single processor servers, trying to provide media servers with schemes for effectively scheduling the available disk bandwidth and storage capacity so that high levels of concurrency and system utilization can be sustained [2, 3]. Current research has been focused on the design of parallel on-demand media servers [4, 1, 11]. A great part of the current research community effort has been also focused on new data layout schemes [9], striping mechanisms, admission control and disk scheduling [11] for storage device arrays of parallel [7] and clustered multimedia servers [6]. The problem of providing deterministic service for VBR streams in a single processor system has been studied in [12] where the notion of the *demand trace* of every VBR stream was introduced.

However, the stream scheduling problem has not been formally addressed. The system resources cannot be fully utilized without a sophisticated scheduling strategy. A very interesting approach described in [8] and [10] provides accurate scheduling of video streams, maximizes system throughput and minimizes the usage of buffers. The main limitation of the method described in [8, 10] is that schedules only CBR video streams with the restrictive assumption that all the streams have been encoded using a unique base stream rate R. Our work is an extension of that scheduling strategy and supports both CBR and VBR media streams (s_is) video and audio encoded using different playback rates (R_is), which is an interesting feature not supported in the original version of the algorithm.

7 Conclusion

This paper is focused on the adaptability and the resource management problems of parallel media servers. A new conflict-free scheduling scheme was presented that provides on-demand support for a large number of concurrent continuous media objects in a predictable manner. The proposed algorithm supports both CBR and VBR encoded media streams video and audio at different playback rates. Using that algorithm, we are able to achieve optimal scheduling in distributed memory parallel architectures. More precisely, our real-time scheduling algorithm guarantees the QoS level of every accepted stream, efficiently utilizes server resources, reduces the required buffer size and increases system throughput. In our adaptability strategy we try to support different quality levels for every connection. The server is able at runtime to allow appropriate media frames to be dropped in a predictable manner without fully suspending service to any one user.

A prototype implementation version of the proposed parallel media server was presented, which demonstrates and confirms the practicability and the efficiency of the resource management strategy in real-life applications.

References

[1] J. Cho and H. Shin. Scheduling video streams in a large-scale video-on-demand server. *Parallel Computing - Special Issue on parallel processing and multimedia*, 23(12):1743–1756, December 1997.

[2] M.N. Garofalakis, B. Özden, and A. Silberschatz. On Periodic Resource Scheduling for Continuous Media Databases. In *Proceedings of the 8th International Workshop on Research Issues in*

Data Engineering, February 1998.

[3] D.J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[4] D. Jadav, C. Srinilta, and A. Choudhary. Batching and dynamic allocation techniques for increasing the stream capacity of an on-demand media server. *Parallel Computing - Special Issue on parallel processing and multimedia*, 23(12):1727–1742, December 1997.

[5] D. Jadav, C. Srinilta, A. Choudhary, and P. Bruce Berra. An evaluation of design tradeoffs in a high performance media-on-demand server. *Multimedia Systems*, 5(1):53–68, 1997.

[6] A. Khaleel and A. Reddy. Evaluation of data and request distribution policies in clustered servers. In *Proceedings of the High Performance Computing*, 1999.

[7] Jack Y.B. Lee. Parallel video servers: A tutorial. *IEEE Multimedia*, 5(2):20–28, 1998.

[8] Chow-Sing Lin, W. Shu, and M.Y. Wu. Performance study of synchronization schemes on parallel cbr video servers. In *Proceedings of the Seventh ACM International Multimedia Conference*, November 1999.

[9] V. Rottmann P. Berenbrink and R. Luling. A comparison of data layout schemes for multimedia servers. In *European Conference on Multimedia Applications, Services, and Techniques (ECMAST'96)*, pages 345–364, 1996.

[10] A. L. Narasimha Reddy. Scheduling and data distribution in a multiprocessor video server. In *Proceedings of the 2nd IEEE Int'l Conference on Multimedia Computing and Systems*, pages 256–263, 1995.

[11] V. Rottmann, P. Berenbrink, and R.I. Lüling. A simple distributed scheduling policy for parallel interactive continuous media servers. *Parallel Computing - Special Issue on parallel processing and multimedia*, 23(12):1757–1776, December 1997.

[12] Ravi Wijayaratne and A. L. Narasimha Reddy. Providing QOS guarantees for disk I/O. *Multimedia Systems*, 8(1):57–68, 2000.

[13] M.Y. Wu and W. Shu. Scheduling for large-scale parallel video servers. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pages 126–133, October 1996.