Architectures and Algorithms for Digital Multimedia On-Demand Servers[*]

P. Venkat Rangan

Multimedia Laboratory
Department of Computer Science and Engineering
University of California at San Diego
La Jolla, CA 92093-0114
E-mail : venkat@cs.ucsd.edu; Phone: (619) 534-5419

Abstract. Future advances in networking coupled with the rapid advances in storage technologies will make it feasible to build multimedia on-demand servers that provide services similar to those of neighborhood videotape rental stores on a metropolitan-area network. A critical requirement in building a multimedia server is the need for guaranteeing continuous playback of media streams. Hence, there are two important questions that need to be addressed in designing a multimedia server: (1) how should media streams be laid out on disk so as to guarantee their continuous retrieval, and (2) how can multiple clients be serviced simultaneously by a multimedia server? In order to address the first question, we propose a constrained block placement policy, in which separations between successive media blocks on disk are bounded so as to guarantee their continuous retrieval at real-time rates. To enable the multimedia server to support multiple clients, we study various policies (such as, round robin and quality proportional) for servicing multiple clients, and propose admission control algorithms for determining whether a new client can be admitted without violating the real-time requirements of any of the clients already being serviced. Finally, we capture the multiplicity of media streams characterizing multimedia objects by defining a multimedia rope abstraction, and describe techniques for their efficient storage on disk, as well as address the problem of servicing multiple rope retrieval requests simultaneously.

1 Introduction

Future advances in networking will make it feasible for computer networks to support digital multimedia transmission. Coupled with the rapid advances in storage technologies, they can be used to build multimedia on-demand services over metropolitan-area networks (such as B-ISDN) that are expected to permeate residential and commercial premises in a manner similar to existing cable TV and telephone networks [12]. A multimedia on-demand server, which we will refer to as a *Multimedia Server* in the rest of this paper, provides services similar to those of a neighborhood videotape rental store. It digitally stores media information such as entertainment movies, educational documentaries, advertisements, etc., on a large array of extremely high-capacity storage

[*] This tutorial is based on work done by the author and his student, Harrick M. Vin, at the UCSD Multimedia Laboratory

devices such as optical or magnetic disks, that are random accessible with a short seek time and are permanently on-line. The multimedia server is connected to *display sites* belonging to clients via a *high-speed network subsystem*. Clients can make a selection of a multimedia object through a variety of indices such as the object's name, and request its retrieval for real-time playback on their display sites. The multimedia server, if it has the necessary resources (such as service time and buffer space), services the client's request by connecting to his/her chosen display site(s), and transmitting the chosen multimedia segment. The retrieval can be interactive, in the sense that clients can stop, pause, resume, and even record and edit the media information if they have permissions to do so. Thus, the multimedia server subsumes the functions of VCRs, videotapes, audio recorders, etc., and can serve varying sizes of clientele: from individual households to entire neighborhoods, and from commercial organizations and educational institutions to national services.

A critical requirement in building a multimedia on-demand service is the need for guaranteeing continuous playback of media streams (since media quanta, such as video frames or audio samples, convey meaning only when presented continuously in time, unlike text in which spatial continuity is sufficient). In order to guarantee continuous playback, (1) the multimedia server must support continuous retrieval from the disk, (2) the network subsystem must guarantee timely delivery of media quanta to the display sites, and (3) the display sites must avoid buffer overruns or starvations. Specifically:

- The multimedia server organizes the storage of media streams in terms of blocks on its disk. In order to guarantee continuous retrieval of a media stream from disk, the multimedia server must constrain the separations between successive media blocks of the media stream so as not to exceed the media blocks' playback durations. Furthermore, servicing multiple clients simultaneously may require that the multimedia server reserve disk access bandwidth for each client prior to rendering the service.
- In order to guarantee timely delivery of media quanta to the display sites, the network subsystem may have to reserve network resources for each client so as to ensure bounds on delay jitter.
- In the absence of variations in playback rates at display sites, ensuring continuous playback at display sites requires that the sites prefetch sufficient number of media quanta, whose total playback duration equals at least the network delay jitter. In the presence of non-deterministic playback rate variations, however, additional mechanisms are essential to enable the multimedia server to detect overruns or starvations of media units at the display sites, and to preventively readjust the transmission rate of media units so as to avoid playback discontinuities.

In this paper, we focus on the first item above, i.e., a multimedia server for supporting continuous retrieval of media streams from disk. The second item, namely the design of a network subsystem for continuous delivery of media quanta has been dealt with by Ferrari and Verma [3]. The last item, i.e., the problem of avoiding overruns and starvations at the display sites in the presence of non-deterministic playback rate variations, has been addressed by Ramanathan and Rangan [7].

Multimedia server designs for guaranteeing continuous retrieval of digital video and audio have, however, remained relatively unexplored. Most of the past work on

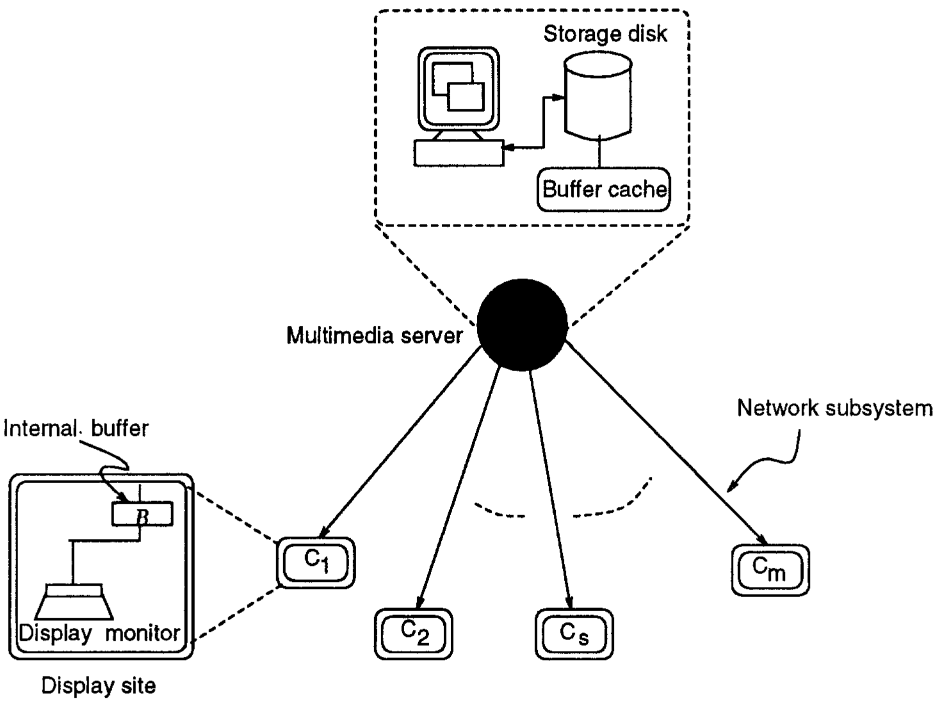

Fig. 1. Configuration of a multimedia on-demand service

multimedia storage systems is restricted to still images and/or audio [1, 5, 6]. Recently, Gammell et al. [4] have described file system designs for supporting audio playback, but they do not address multi-user video on-demand services. A qualitative design for a file system offering video services is presented by the author in [8]. A quantitative model for the design of a file system for storing real-time video and audio streams on disks have been presented by Rangan and Vin [10, 11]. Admission control algorithms for multi-user video on-demand servers are presented by Vin and Rangan [13]. A review of these architectures and algorithms for designing a high-performance, multimedia server capable of servicing a large number of clients is the subject matter of this paper.

First, we address the problem of storing multiple media *strands*, each of which denotes a sequence of continuously recorded video frames or audio samples. We propose a constrained block allocation policy, in which separations between successive media blocks on disk are bounded so as to guarantee their continuous retrieval at real-time rates.

Then, we address the question of servicing multiple clients by a multimedia server. Given the maximum rate of disk data transfer, the multimedia server can only service a limited number of clients simultaneously. We study various policies (such as, round robin and quality proportional) for servicing multiple clients, and propose *admission control algorithms* for determining whether a new client can be admitted without violating the real-time requirements of any of the clients already being serviced. In the quality proportional servicing (QPMS) algorithm proposed in this paper, the number of media blocks of a strand retrieved during each service round is proportional on an average to the the strand's playback rate, and successive number of media blocks retrieved are fine tuned individually to achieve the servicing of an optimal number of clients.

We capture multiplicity of media streams constituting a multimedia object by defining a *multimedia rope* abstraction, which represents a collection of media strands tied together by synchronization information. We propose a merging algorithm for efficient storage and retrieval of media strands constituting a rope, and describe techniques for servicing multiple rope retrieval requests simultaneously.

The rest of this paper is organized as follows: Constrained placement policies are developed in Section 2. The admission control algorithm is described in Section 3. Merged storage and retrieval of ropes are addressed in Section 4. Section 5 presents performance evaluation, and finally, Section 6 concludes the paper.

2 Managing Storage of Digital Multimedia

Digitization of video yields a sequence of frames, and that of audio yields a sequence of samples. We refer to a sequence of continuously recorded video frames or audio samples as a *Strand*. A multimedia server must divide video and audio strands into blocks while storing them on a disk. Continuous playback of media strands requires that the time for retrieving a media block of a strand from disk does not exceed the media block's playback duration.

Most existing storage server architectures employ unconstrained placement of blocks on disk. In such storage servers, reserving computational cycles to meet real-time requirements is not sufficient to support continuous retrieval of media strands. This is because, separations between blocks of a strand may not be constrained enough to guarantee bounds on seek and rotational latencies incurred while accessing successive blocks of the strand. Contiguous placement of media blocks, on the other hand, guarantees that successive blocks can be retrieved without incurring any seek or rotational latency. However, contiguous placement of media blocks is fraught with inherent problems of fragmentation, and can entail enormous copying overheads during insertions and deletions.

Constrained block placement maintains the access time of media blocks within the real-time playback requirements of strands by bounding the separation between successive media blocks on disk. Even the projected speeds of future fast disk configurations are not sufficient to ensure that unconstrained separations between blocks lie within the requirements of high performance video applications. Hence, constrained block placement is not an artifact of today's storage performance, but a fundamental problem that is not likely to be obviated by the availability of faster storage devices in the near future.

There are two questions that need to be answered in constrained placement of media blocks on disk: (1) What should the size of the blocks (i.e. the *granularity*) be? and (2) What should the separation between successive blocks (i.e. the *scattering*) of a strand be? Together, they define the *storage pattern* of a media strand (see Figure 2). Whereas granularity can be determined using the available buffer space at display sites, upper and lower bounds on scattering can be derived using the requirements of continuous playback, and maximizing the data transfer rate, respectively. In this section, we determine these parameters for digital video (which is the most demanding medium with respect to performance and storage space requirements); the analysis for audio can be carried out in a similar manner.

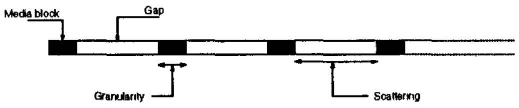

Fig. 2. Storage pattern of a media strand

2.1 Determining Granularity and Scattering

During playback, media blocks are transmitted by a multimedia server to display sites belonging to clients. Consequently, the sizes of internal buffers available at the display sites can be used to determine granularity. For instance, if internal buffers available at display sites can store multiple video frames (say f), then the buffers can be partitioned into two sets (each capable of holding $f/2$ frames): one set to hold the blocks being transmitted by the multimedia server, and another set to hold the blocks being displayed. Hence, each media block may contain $f/2$ frames, yielding $\eta_{vs} = f/2$.

Symbol	*Explanation*	*display unit*
$\mathcal{R}_{vp}$	Video playback rate	frames/sec
$\mathcal{R}_{dr}$	Disk data transfer rate	bits/sec
η_{vs}	Granularity of video storage	frames
s_{vf}	Size of a video frame	bits/frame
l_{ds}^{l}	Lower bound on scattering	sec
l_{ds}^{u}	Upper bound on scattering	sec

Table 1. Symbols used in this paper

The guiding factor in determining the upper bound on scattering is the requirement of continuous playback. Whereas playback durations of media blocks of a strand depend on the playback rate of the strand, the time for retrieving a sequence of blocks is a function of their placement on disk. Table 1 defines the symbols used for the parameters governing continuity requirements, using which, it can be seen that the playback duration of a media block is given by $\frac{\eta_{vs}}{\mathcal{R}_{vp}}$. Continuous playback at the media playback rate requires that the time to access each media block from disk (given by $l_{ds}^{u} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}$) be bounded by its playback duration, yielding:

$$l_{ds}^{u} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}} \leq \frac{\eta_{vs}}{\mathcal{R}_{vp}} \tag{1}$$

which we refer to as the *continuity equation.* Thus, having determined the granularity, the upper bound on scattering l_{ds}^{u} can be determined by direct substitution in the continuity equation.

Even though bounding the separation between successive media blocks so as not to exceed l_{ds}^{u} ensures continuous retrieval of media strands, the value of l_{ds}^{u} derived from the continuity equation may, in general, be significantly larger than the time to read a media block from disk (namely, $\frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}$). Hence, if a placement policy is based solely on l_{ds}^{u}, then only a small fraction of the time required to access a media block may be spent in reading its contents from disk, thereby yielding low data transfer rates. In order to maximize the data transfer rate, it is essential that media blocks be placed on disk in a *rotationally optimal* manner. The rotationally optimal separation between media blocks depends on the characteristics of the multimedia server (such as, the delay incurred in initiating a new disk block access after having completed a previous request). If the separation between successive media blocks on disk is smaller than rotationally optimal separation, then while accessing each pair of successive media blocks, the disk head may go past the location on disk containing the next media block after having retrieved the previous media block, before the next read operation read can be initiated. Consequently, maximum rotational latency may be incurred in accessing the next media block. Thus, rotationally optimal separation defines a lower bound on scattering between successive media blocks, and is denoted by l_{ds}^{l}.

2.2 Constrained Placement of Media Strands

Consider the problem of placing a media strand on disk. Assume that the storage space of the disk is divided into tracks, each track is subdivided into several disk blocks, and accessing a disk block requires positioning the disk head on the track containing the disk block (thereby incurring seek latency), and then waiting for the block to rotate under the disk head (thereby incurring rotational latency). The seek time is assumed to vary linearly with the seek distance (expressed in terms of number of tracks), and the maximum rotational latency is assumed to be bounded by l_{rot}^{max}. The goal of the constrained placement algorithm is to allocate disk blocks to media blocks such that the separation between successive media blocks on disk conforms to the bounds on scattering.

Specifically, given that a media block B_i of a strand S, with bounds on scattering $[l_{ds}^{l}, l_{ds}^{u}]$, is placed in disk block d on track t, the algorithm determines disk block d_{new} on track t_{new} for storing media block B_{i+1} of strand S, such that the seek and rotational latencies incurred while moving the disk head from media block B_i to B_{i+1} is within $[l_{ds}^{l}, l_{ds}^{u}]$. Once a disk head is positioned on track t_{new}, since any block on that track can be retrieved within time l_{rot}^{max}, the feasibility of storing media block B_{i+1} on track t_{new} depends on $(l_{ds}^{l} - l_{seek})$ and $(l_{ds}^{u} - l_{seek})$, where l_{seek} denotes the time to seek from track t to t_{new}. We refer to $(l_{ds}^{l} - l_{seek})$ and $(l_{ds}^{u} - l_{seek})$ as the *residual lag time* (r_{lag}) and *residual slack time* (r_{slack}), respectively. Clearly, if $r_{slack}, r_{lag} > l_{rot}^{max}$ or if $r_{slack}, r_{lag} < 0$, then it is not possible to place block B_{i+1} on track t_{new}. In all other cases, the disk blocks on track t_{new} can be partitioned into feasible and infeasible sets such that allocating any disk block from the feasible set guarantees that the separation between media blocks B_i and B_{i+1} is within $[l_{ds}^{l}, l_{ds}^{u}]$. None of the blocks from the infeasible set can be allocated to B_{i+1}, and hence, remain available for allocation to future media blocks. Thus, given the disk characteristics (namely, a, b, and l_{rot}^{max}), as well as the strand characteristics (namely, l_{ds}^{l} and l_{ds}^{u}), the constrained placement algorithm determines a track and a disk block within that track where media block B_{i+1} can be stored.

A *strict* placement algorithm guarantees that the separation between each pair of successive media blocks is within $[l_{ds}^{l}, l_{ds}^{u}]$ (i.e., $r_{lag} < 0$ and $r_{slack} > 0$). On the contrary, an *adaptive* placement algorithm may accommodate occasional violations of the bounds on scattering (yielding $r_{lag} > 0$ or $r_{slack} < 0$), as long as the average separation between successive media blocks over a finite window of blocks is within $[l_{ds}^{l}, l_{ds}^{u}]$. Whereas strict placement of a media strand on disk permits its playback to be initiated from an arbitrary block without any read-ahead, an adaptive placement may require a read-ahead equal to the number of media blocks within an averaging window. The adaptive placement algorithm, however, is much more flexible since it may succeed in placing media blocks on disk even when the strict algorithm fails to do so.

3 Servicing Multiple Clients Simultaneously

Till now, we have investigated techniques for placing a media strand on disk so as to guarantee its continuous retrieval in isolation. However, in practice, a multimedia server has to process requests from several clients simultaneously. In the best scenario, all the clients may request the retrieval of the same media strand, in which case, the multimedia server needs only to retrieve the strand once from the disk and then multicast it to all the clients. However, more often than not, different clients may request the retrieval of different strands; and even when the same strand is being requested by multiple clients (such as a popular movie), there may be phase shifts among their requests (e.g., each client viewing a different part of the movie at the same time). A simple mechanism to guarantee that the real-time requirements of none of the clients are violated is to dedicate a disk head to each client, which, however, limits the total number of clients to the number of disk heads. On the other hand, if the data transfer rate of the disk is higher than the requirements of a single client, then the number of clients that can be serviced simultaneously can be significantly increased by multiplexing a disk head among several clients. However, given the maximum rate of disk data transfer, the multimedia server can only service a limited number of clients. Hence, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted without violating the continuity requirements of any of the clients already being serviced.

3.1 Formulating the Admission Control Problem

Continuous playback of a media strand involves a sequence of periodic tasks with deadlines, where tasks correspond to retrievals of media blocks from disk, and deadlines correspond to the scheduled playback times of media blocks. Thus, servicing multiple strand retrieval requests requires the derivation of a real-time schedule, for which the complexity of the best known algorithms show quadratic dependence on the number of tasks. Since strands usually consist of a large number of media blocks (e.g., if each media block contains one video frame, then a five minute clip of a HDTV video strand recorded at 60 frames/s contains 18000 blocks), the number of tasks can be very large. Hence, direct application of traditional real-time scheduling techniques is out of question.

Consider a multimedia server that is required to concurrently service requests for strands $S_1, S_2, ..., S_n$. Since each request is periodic, the multimedia server can service them by proceeding in *rounds*. Suppose that, during each round, the multimedia server retrieves a sequence of k_1 media blocks of strand S_1, and k_2 media blocks of strand S_2, ..., and k_n media blocks of strand S_n. The total time required to complete the round should not exceed the minimum of the playback durations of k_1, k_2, ..., or k_n blocks. Whereas the playback duration of a sequence of media blocks of a strand is a function of the playback rate of that strand, the retrieval rate of media blocks is a function of their placement on disk. Thus, the policies for servicing multiple clients can be classified into two main categories: *deadline based* and *placement based*. Whereas the former retrieves media blocks based on the earliest deadline first scheduling policy, the latter retrieves media blocks from disk so as to minimize the total seek and rotational latencies incurred during retrieval. Servicing policies can be applied either to the media blocks within a strand (yielding a *local schedule*) or the global pool of media blocks from all the strands (yielding a *global schedule*). Clearly, when servicing policies are applied among media blocks within a strand, the multimedia server has to employ ordering techniques (such as, round robin ordering) to switch from one strand to next during each round. We will now formulate the problem of servicing multiple strand retrieval requests assuming a deadline based servicing policy for deriving local schedules and round robin ordering of strands, and describe an admission control algorithm which a multimedia server can employ to decide whether a new client request can be admitted without violating the real-time requirements of the clients already being serviced.

Let us suppose that a multimedia server is servicing n client, each retrieving a different media strand (say, $S_1, S_2, ..., S_n$, respectively). Let $\eta^1_{vs}, \eta^2_{vs}, \ldots, \eta^n_{vs}$ denote the granularities of the n strands being retrieved, $l^1_{ds}, l^2_{ds}, \ldots, l^n_{ds}$ denote the upper bounds on scattering, and $\mathcal{R}^1_{vp}, \mathcal{R}^2_{vp}, \ldots, \mathcal{R}^n_{vp}$ their playback rates. Assuming round-robin ordering of strands, the multimedia server retrieves a finite number of media blocks k_i of each strand S_i, $i \in [1, n]$ in accordance with the earliest deadline first policy, before switching to the next strand. Whereas the rate of transfer of successive blocks of a strand is governed by its granularity and scattering, switching from one strand to another may entail an overhead of up to the maximum seek and rotational latencies (since the layout does not constrain the relative positions of two different strands). The continuity requirement for each strand can be satisfied if and only if the service time per round does not exceed the minimum of the playback durations of k_1, k_2, ..., or k_n blocks. That is,

$$n * (l^{max}_{seek} + l^{max}_{rot}) + \sum_{i=1}^{n} \sum_{j=1}^{k_i - 1} \left(l^i_{ds} + \frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}} \right) \leq \min_{i \in [1,n]} \left(k_i * \frac{\eta^i_{vs}}{\mathcal{R}^i_{vp}} \right) \qquad (2)$$

Clearly, evaluating the validity of Equation (2) for each round, using the precise values of media block sizes and the separation between successive media blocks for each strand, is computationally infeasible. Hence, in order to provide *deterministic* service guarantees to each of the n clients, the values of l^i_{ds} and s^i_{vf}, in Equation (2), $\forall i \in [1, n]$, must be set to their respective maximum values. However, this may be very pessimistic, since constrained block placement algorithm and variable rate compression techniques (such as, JPEG and MPEG) may yield l_{ds} and s_{vf} significantly smaller than their respective maximum values. Consequently, the multimedia server can service a larger number of clients by exploiting the variable reductions in l_{ds} and s_{vf}, and providing *statistical* service guarantees to each of the clients. Specifically, if l^i_{ds} and s^i_{vf} represent random variables characterizing the separation between successive media blocks, and the bit size distribution of frames yielded by compression techniques such as JPEG and MPEG, respectively, then the term

$$\sum_{i=1}^{n} \sum_{j=1}^{k_i - 1} \left(l^i_{ds} + \frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}} \right)$$

in Equation (2) represents the sum of $2 * \sum_{i=1}^{n}(k_i - 1)$ independent random, and can be denoted as a random variable χ. Hence, Equation (2) reduces to:

$$\chi \leq \min_{i \in [1,n]} \left(k_i * \frac{\eta^i_{vs}}{\mathcal{R}^i_{vp}} \right) - n * \left(l^{max}_{seek} + l^{max}_{rot} \right) \tag{3}$$

If F_χ is the distribution function of χ, then guaranteeing continuous playback of n video strands with a probability greater than π necessitates that:

$$F_\chi \left(\min_{i \in [1,n]} \left(k_i * \frac{\eta^i_{vs}}{\mathcal{R}^i_{vp}} \right) - n * \left(l^{max}_{seek} + l^{max}_{rot} \right) \right) \geq \pi \tag{4}$$

The multimedia server can service all the n clients simultaneously if and only if $k_1, k_2, ..., k_n$ can be determined such that either Equation (2) (in the case of deterministic guarantees) or Equation (4) (in the case of statistical guarantees) is satisfied. Since both of these formulations contain n parameters and only one equation, determination of the values of $k_1, k_2, ..., k_n$ require additional techniques. The simplest technique for the choice of $k_1, k_2, ..., k_n$ is to use the same value for all of them, yielding what is generally referred to as a *round robin servicing algorithm with fixed quanta*. However, this certainly may not be the optimal number of clients, because, whereas the strand with the maximum playback rate will have retrieved exactly the number of media blocks it needs for the duration of a service round, other strands with smaller playback rates will have retrieved more media blocks than they need in each service round (thereby, leading to accumulation of media blocks at display sites). Consequently, by reducing the number of media blocks retrieved per service round for such strands, it may be possible to accommodate more number of clients. We now propose a *quality proportional multi-client servicing algorithm* that allocates values to k_i proportional to the playback rate of the strand S_i, and is guaranteed to yield values of k_i so as to satisfy Equation (2) whenever a solution exists for the given number of clients.

3.2 Quality Proportional Multi-client Servicing

In the *Quality Proportional Multi-client Servicing (QPMS)* algorithm, the number of blocks accessed during each round for each strand is proportional to its playback rate. That is,

$$\forall i \in [1, n] : \quad k_i \propto \mathcal{R}^i_{vp}$$

If k is the proportionality constant, using which, we get, $k_1 = k * \mathcal{R}_{vp}^1$, $k_2 = k * \mathcal{R}_{vp}^2$, ..., $k_n = k * \mathcal{R}_{vp}^n$. Under these conditions, Equation (2) reduces to:

$$n*(l_{seek}^{max}+l_{rot}^{max})+k*\sum_{i=1}^{n}\mathcal{R}_{vp}^i*(l_{ds}^i+\frac{\eta_{vs}^i * s_{vf}^i}{\mathcal{R}_{dr}})-\sum_{i=1}^{n}(l_{ds}^i+\frac{\eta_{vs}^i * s_{vf}^i}{\mathcal{R}_{dr}}) \leq k*\eta_{vs} \quad (5)$$

Given the granularity and scattering parameters for each strand, Equation (5) can be used to determine k, from which, the number of blocks retrieved during each service round can be obtained as: $k_1 = k*\mathcal{R}_{vp}^1$, $k_2 = k*\mathcal{R}_{vp}^2$, ..., $k_n = k*\mathcal{R}_{vp}^n$. It can be shown that this algorithm always yields values of k_i so as to satisfy Equation (2) whenever a solution exists for the given number of clients [13].

Notice, however, that the values of k_i's obtained using the QPMS algorithm may not be integral. Since the display of media strands proceeds in terms of quanta such as frames, if k_i is not an integer, then retrieval of a fraction of a frame cannot be used for display, causing the display to starve until the remaining fraction arrives, possibly in the next service round. Such scenarios can be avoided if k_i's are all integers, techniques for deriving which we now elaborate, starting from the real values yielded by the QPMS algorithm.

Let the values of $\{k_1, k_2, ..., k_n\}$ yielded by the QPMS algorithm be given by: $\forall i \in [1, n]: \quad k_i = I_i + F_i$, where I_i and F_i are the integer and the fractional parts of k_i, respectively. If $I = \sum_{i=1}^{n} I_i$ and $F = \sum_{i=1}^{n} F_i$, then $(I + F)$ denotes the average number of blocks that need to be retrieved in each service round. In the technique that we present, the number of blocks of strand S_i retrieves during a service round toggles between $\lfloor k_i \rfloor$ and $\lceil k_i \rceil$, so that on an average, the transfer rate for each strand S_i is k_i blocks/round. Specifically, for each round r, the multimedia server must determine the set $\mathcal{K}^r = \{k_1^r, k_2^r, \cdots k_n^r\}$ of the sequence of number of blocks of the n strands to be retrieved during round r, where k_i^r can equal either $\lfloor k_i \rfloor$ or $\lceil k_i \rceil$. However, in doing so, both the service time and buffer space constraints, that would have been met had k_i blocks been retrieved for every round, must continue to be satisfied. Maintenance of the continuity requirement requires that the cumulative slack time at the multimedia server, which is the sum of the differences between the RHS and the LHS of Equation (2) for each round, must be non-negative so as to ensure that none of the clients are starved during a service round. Similarly, the buffer space constraint requires that the slack buffer space at the multimedia server, which is the difference between the available buffer space and the used buffer space, must be non-negative. To ensure that both the constraints are not violated, the toggling of $\lfloor k_i \rfloor$ to $\lceil k_i \rceil$ for strands must be dynamically staggered. The order of toggling can be determined as follows:

Since during every round, k_i blocks of strand S_i are consumed on an average, during rounds in which $\lfloor k_i \rfloor$ blocks are retrieved, there must be sufficient accumulation of data at display sites belonging to clients to maintain continuity of playback, and the accumulation is resumed during rounds in which $\lceil k_i \rceil$ blocks are retrieved. Furthermore, an initial prefetching of blocks is also necessary to guarantee continuity during the first few rounds (since not all strands S_i can have $\lceil k_i \rceil$ blocks retrieved during the first few rounds). Thus, the accumulation at the end of round R for client i is the sum of differences between k_i^r and k_i during the R rounds plus the prefetched number of blocks

$\mathcal{P}_i$, and is given by:

$$\mathcal{D}_i(R) = \mathcal{P}_i + \sum_{r=1}^{R}(k_i^r - k_i) \tag{6}$$

During a round R, if $\mathcal{D}_i(R) < F_i$, a shortage of blocks would occur during the next round; hence, round R is the *deadline* for accessing $\lceil k_i \rceil$ blocks of strand S_i. During each round, if there is sufficient slack time available to transfer extra blocks, strands are ordered with earliest deadline round first, and $\lceil k_i \rceil$ blocks are retrieved for each such strand S_i until the exhaustion of the slack time. During each service round, if for all the strands, k_i^r is set to I_i, then the multimedia server can retrieve the extra blocks of at least $\lfloor \sum_{i=1}^{n}(k_i - I_i) \rfloor = \lfloor F \rfloor$ strands in the order of earliest occurring deadline first, and whenever sufficient slack time accumulates, retrieve the extra blocks of $\lceil F \rceil$ strands. Such a policy allows the deadline requirements of the maximum number of strands to be satisfied as much in advance as possible, while at the same time limiting the maximum extra buffering needed during each round to $\lceil F \rceil$.

4 From Media Strands to Multimedia Ropes

A multimedia object consists of several media components (such as, audio and video). We refer to a collection of media strands tied together by synchronization information as a *multimedia rope*. Synchronization information among media strands constituting a rope can be expressed by relating the playback intervals of media strands in one of thirteen possible ways [2]: *before, meets, overlaps, during, starts, ends, equals,* plus the inverse relations - except *equals*. In this section, we first describe techniques for efficient storage and retrieval of multimedia ropes on disk, and then address the problem of servicing multiple rope retrieval requests.

4.1 Efficient Storage of Multimedia Ropes

Consider a multimedia rope M consisting of strands $S_1, S_2, ...,$ and S_n. A straightforward approach for storing these strands is to permit each disk block to contain media samples from various strands (i.e., *heterogeneous* blocks). For instance, if S_1 and S_2 denote a video and an audio strand, respectively, then a video frame and corresponding audio samples can be stored in the same disk block. Whereas such a storage scheme affords the advantage that it provides implicit inter-media synchronization, it entails additional processing for combining these media during storage, and for separating them during retrieval.

A better approach is to restrict each disk block to contain exactly one medium (i.e., *homogeneous* blocks). Such a scheme permits the multimedia server to exploit the properties of each medium to independently optimize the storage of each media strand. However, the multimedia server must maintain explicit relationships among the playback intervals of strands so as to ensure their synchronous retrieval.

Using homogeneous blocks, a simple scheme for storing a multimedia rope is to independently layout blocks of each of its constituent strands. However, playback of such a rope may incur significant seek and rotational latencies while concurrently accessing media blocks of its constituent strands. Since media blocks of all the strands constituting a rope may be concurrently available at the time of storage, the multimedia server can minimize the overhead due to seek and the rotational latencies incurred during retrieval, due to switching between strands, by filling up the gaps between media blocks of one strand with media blocks of other strands. We refer to the process of storing media blocks of a strand in the gaps between successive blocks of other strands as *merging*.

Intuitively, the storage of n strands $S_1, S_2, ..., S_n$ can be merged together if the sum of the fractions of space occupied by their media blocks does not exceed 1. Thus, if $\eta^1_{ms}, \eta^2_{ms}, \ldots, \eta^n_{ms}$ denote the granularities, and $l^1_{ds}, l^2_{ds}, \ldots, l^n_{ds}$ denote the upper bounds on scattering, for strands S_1, S_2, ..., S_n, respectively, then the condition for merging their storage can be formally stated as:

$$\sum_{i=1}^{n} \left(\frac{\eta^i_{vs} * s^i_{vf}}{\eta^i_{vs} * s^i_{vf} + l^i_{ds} * \mathcal{R}_{dr}} \right) \leq 1 \tag{7}$$

where $\eta^i_{vs} * s^i_{vf}$ and $(\eta^i_{vs} * s^i_{vf} + l^i_{ds} * \mathcal{R}_{dr})$ denote the sizes (in terms of bits) of media blocks and storage pattern of strand S_i, respectively.

Suppose that media strands are placed on disk such that chunks of k_1 blocks of S_1, k_2 blocks of S_2, ..., and k_n blocks of S_n follow each other, and the sequence repeats (see Figure 3). Consequently, guaranteeing retrieval of each strand S_i at its playback rate requires that the space occupied by blocks of all the other strands S_j $(j \neq i)$, between two successive chunks of blocks of S_i, does not exceed the total gap space permitted for k_i blocks (present in each chunk) of S_i. That is,

$$\forall \text{ strands } S_i, i \in [1, n] : \quad \sum_{j \in [1,n], j \neq i} k_j * \left(\eta^j_{vs} * s^j_{vf}\right) \leq k_i * \left(l^i_{ds} * \mathcal{R}_{dr}\right) \tag{8}$$

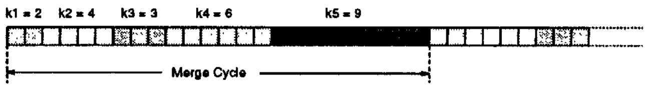

Fig. 3. Merged storage of media strands

The values of $k_1, k_2, \ldots, k_n$ satisfying the above system of n equations define a *merge cycle*. As a solution to the above system of equations, we now propose a *scaled placement policy*, in which the number of consecutive blocks k_i of a strand S_i placed in a merge cycle is inversely scaled by the length of its storage pattern (i.e., $\eta^i_{vs} * s^i_{vf} + l^i_{ds} * \mathcal{R}_{dr}$). That is, $\forall i \in [1, n]$:

$$k_i = \frac{k}{\left(\eta^i_{vs} * s^i_{vf} + l^i_{ds} * \mathcal{R}_{dr}\right)} \tag{9}$$

where, k is a constant. Substituting the values of k_i's obtained from Equation (9), and rearranging the terms of Equation (8), it can be shown that the scaled placement policy is guaranteed to yield a merge cycle whenever the merge condition (Equation (7)) is satisfied [9].

When $k_1, k_2, ..., k_n$ in a merge cycle satisfy Equation (8), for each strand S_i, fetching its k_i blocks within each merge cycle is sufficient to guarantee continuous retrieval for the duration of the merge cycle. Hence, at a display site, up to $2 * k_i$ buffers may be required for strand S_i: one set of k_i buffers to hold the blocks being transferred, and another set to hold the blocks being displayed. In turn, given the bounds on buffering available at display devices (which is in fact the case in most hardware environments), bounds on the values of k_i can be fixed, from which, bounds on the values of k can be determined by Equation (9). Among all such bounds of k, the lowest is chosen as the value of k, from which the tightest values of k_i are recomputed, again by using Equation (9).

Notice, however, that the values of k_i's so obtained may not be integral (unless k is chosen to be an integral multiple of the LCM of the storage pattern lengths, which, of course, can be very large). In order to ensure continuous retrieval of media strands, the values of k_i's must be integral. By using a technique similar to one presented in Section 3.2 for the QPMS algorithm, the integral number of media blocks required to be stored in each merge cycle can be derived by toggling between $\lfloor k_i \rfloor$ and $\lceil k_i \rceil$ for each strand in a staggered manner between successive merge cycles, so that on an average, the number of blocks of strand S_i stored in a merge cycle equals k_i.

4.2 Admission Control Algorithm for Multimedia Ropes

Playback of a multimedia rope may require simultaneous or sequential display of its constituent media strands. Hence, the data transfer requirement of a rope may vary during its playback. The admission control algorithms described in Section 3 have assumed a fixed data transfer requirement for each strand throughout the duration of its playback. In this section, we present a technique for partitioning the playback duration of a rope into intervals, each with fixed data transfer requirements, thereby reducing the problem of servicing a rope retrieval request to a set of problems for servicing multiple strand retrieval requests.

Given the relationship between the playback intervals of media strands, the data transfer requirement of a multimedia rope can be completely characterized by maintaining: (1) the time instants at which the playback of its constituent media strands begin and end during the playback of the rope, and (2) the extent of increase or decrease in the data transfer requirement. Formally, for a multimedia rope M_l, we define *alteration points* (denoted by a_l^i) as the time instants at which the playback of its constituent strands either begin or end. We refer to an ordered set (sorted in the increasing order of time) of alteration points as an *alteration set*, and denote it by $\mathcal{A}_l$. We refer to the time interval between successive pairs of alteration points (namely, $\forall i \in [1, n_l] : [a_l^i, a_l^{i+1}]$ where $n_l = |\mathcal{A}_l|$) as an *alteration interval*. Since each alteration interval may involve simultaneous playback of multiple strands, the data transfer requirement for each alteration interval can be represented as a set of the data transfer requirements of strands (defined by the 4-tuple $\{\eta_{vs}, s_{vf}, l_{ds}, \mathcal{R}_{vp}\}$), and is referred to as the *playback set* (denoted by ψ^i). Thus, the data transfer requirement of a rope M_l can be uniquely represented as a pair $\{\mathcal{A}_l, \Psi_l\}$, where $\mathcal{A}_l$ denotes the alteration set and Ψ_l denotes a sequence of playback sets.

Consider the process of initiating simultaneous playback of multimedia ropes M_1 and M_2. Let the data transfer requirements of ropes M_1 and M_2 be characterized by $\{\mathcal{A}_1, \Psi_1\}$ and $\{\mathcal{A}_2, \Psi_2\}$, respectively. Let $|\mathcal{A}_1| = n_1$ and $|\mathcal{A}_2| = n_2$, and let

$$\mathcal{A}_1 = \{a_1^1, a_1^2, ..., a_1^{n_1}\}$$
$$\mathcal{A}_2 = \{a_2^1, a_2^2, ..., a_2^{n_2}\}$$

Similarly, let

$$\Psi_1 = \{\psi_1^0, \psi_1^1, ..., \psi_1^{n_1-1}, \psi_1^{n_1}\}$$
$$\Psi_2 = \{\psi_2^0, \psi_2^1, ..., \psi_2^{n_2-1}, \psi_2^{n_2}\}$$

where $\psi_1^0 = \psi_1^{n_1} = \psi_2^0 = \psi_2^{n_2} = \emptyset$. If the playback of ropes M_1 and M_2 are initiated simultaneously, then the data transfer requirement will change at each of the alteration points of M_1 and M_2. Consequently, the cumulative alteration set $\mathcal{A} = \{a^1, a^2, ..., a^{n_1+n_2}\}$ can be obtained by performing a *merge sort* operation on $\mathcal{A}_1$ and $\mathcal{A}_2$. Furthermore, simultaneous playback of ropes M_1 and M_2 yields at most (n_1+n_2+1) alteration intervals, the playback set ψ^i for each interval can be determined using an *iterative algorithm*:

- If $\exists j_1 \in [1, n_1]$ such that $a^i = a_1^{j_1}$, then

$$\psi^i = (\psi^{i-1} - \psi_1^{j_1-1}) \cup \psi_1^{j_1}$$

- If $\exists j_2 \in [1, n_2]$ such that $a^i = a_2^{j_2}$, then

$$\psi^i = (\psi^{i-1} - \psi_2^{j_2-1}) \cup \psi_2^{j_2}$$

Thus, playback of ropes M_1 and M_2 can be initiated simultaneously if and only if the multimedia server can satisfy the data transfer requirements of each of the cumulative alteration intervals. If, however, the multimedia server is unable to meet the data transfer requirement of any one of the intervals, the earliest time instant at which the playback of M_2 can be initiated, given that the playback of M_1 has already been scheduled, can be determined by delaying the initiation of M_2 by an alteration interval of M_1, and repeating the analysis.

5 Experience and Performance Evaluation

A prototype multimedia server is being implemented at the UCSD Multimedia Laboratory in an environment consisting of multimedia stations connected to a multimedia server through Ethernet and FDDI networks. Each multimedia station consists of a computing workstation, a PC-AT, a video camera, and a TV monitor. The PC-ATs are equipped with digital video processing hardware that can digitize and compress motion video at real-time rates, and audio hardware that can digitize voice at 8 KBytes/sec. The multimedia server is implemented on a 486-PC equipped with multiple gigabytes of storage.

The software architecture of the prototype multimedia server consists of two functional layers: the *Storage Manager* and the *Rope Server* (see Figure 4). The storage manager is responsible for physical storage of strands on disk, and handles determination of granularity and scattering parameters for strands, constrained placement of media blocks on disk, and merged storage of strands constituting a rope. The rope server, on the other hand, handles maintenance of synchronization relationships between strands, and admission control. The rope server also provides facilities for users to create, edit, and retrieve multimedia ropes.

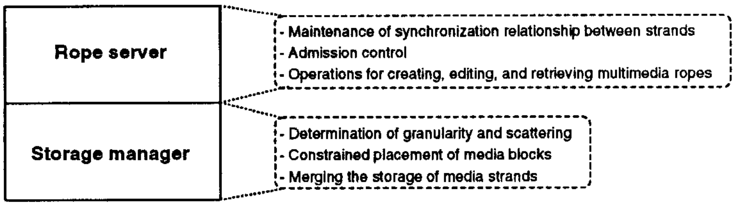
Rope server
Storage manager
- Maintenance of synchronization relationship between strands
- Admission control
- Operations for creating, editing, and retrieving multimedia ropes
- Determination of granularity and scattering
- Constrained placement of media blocks
- Merging the storage of media strands

Fig. 4. Software architecture of the prototype multimedia server

We have carried out simulations to evaluate the performance of various media block placement policies. Our simulations have shown that the data transfer bandwidths yielded by both unconstrained and constrained placement policies improve with increase in disk block size. This is because, increasing the disk block size results in a reduction in the number of disk blocks required to store a media strand, thereby reducing the total seek and rotational latency overhead. However, even at large disk block sizes, unconstrained placement policy can achieve only about 3% of the maximum data transfer bandwidth. The performance of the constrained placement policy, on the other hand, depends on the average separation between successive media blocks. As the average separation approaches l^l_{ds} (derived using the rotationally optimal separation), the data transfer bandwidth yielded by the constrained placement policy approaches the maximum data transfer bandwidth of the disk.

We have also evaluated the relative performance various deterministic and statistical admission control algorithms. Our analysis demonstrates that providing statistical service guarantees to video strands encoded using JPEG or MPEG compression techniques yields smaller values of k_i's (and hence, imposes smaller buffer space requirement), and can service a larger number of clients simultaneously, as compared to its deterministic counterpart (see Figure 5). These experiments also illustrated the gain in the maximum number of simultaneous clients in the QPMS as compared to the round-robin algorithm. Higher the asymmetry among the playback rates of the client requests, greater is the advantage of employing the QPMS algorithm. When the playback rates of all the clients are the same, the performance of the QPMS algorithm degenerates to that of the round-robin algorithm.

6 Concluding Remarks

Constrained placement of media blocks on disk does not entail the disadvantages of contiguous and unconstrained placement policies, and ensures that the access time of

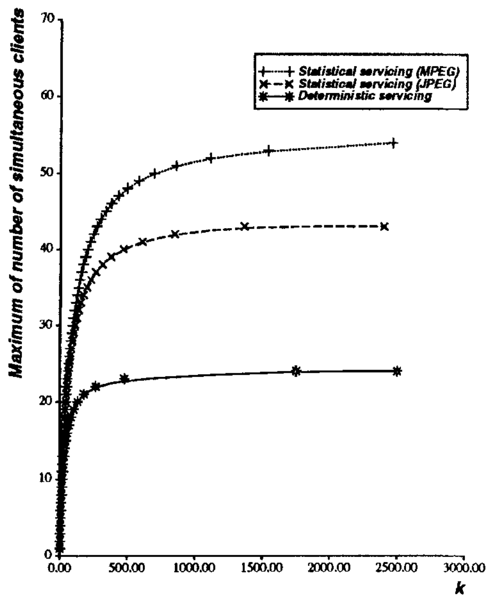

Fig. 5. Relative variations in the number of clients (n) that can be serviced with the length of a service round (in number of media blocks k) for deterministic servicing and statistical servicing (JPEG and MPEG) of video requests.

media blocks within the real-time playback requirements of strands. However, multimedia servers employing constrained placement policy may be required to fill gaps between media blocks of one strand with media blocks of other strands, so as to utilize the storage space efficiently.

Whereas constrained placement of a media strand can only guarantee its continuous retrieval in isolation, a multimedia server, in practice, has to service multiple clients simultaneously. Given the maximum rate of disk data transfer, the multimedia server can only service a limited clients without violating the continuity requirements of any one of them. The admission control algorithm depends on: (1) the real-time requirements imposed by each client, (2) the type of service rendered by the multimedia server (i.e., deterministic or statistical), (3) the servicing policy (namely, deadline based or placement based), and (4) whether the servicing policy is applied to media blocks within a request (yielding a local schedule) or to the global pool of media blocks from all the requests (yielding a global schedule).

We have studied several policies for (such as, round robin and quality proportional) for servicing multiple clients, and have proposed algorithms by which a multimedia server can enforce these policies without violating the real-time retrieval rates of any of the clients. The quality proportional servicing algorithm retrieves media blocks at a rate proportional on an average to the media playback rates of requests, but uses a staggered toggling technique by which successive numbers of retrieved media blocks are fine tuned individually to achieve the servicing of an optimal number of clients. Our performance analysis illustrates that the constrained placement policy achieves significantly higher effective data transfer bandwidth as compared to unconstrained placement of media strands, and the QPMS algorithm for servicing multiple clients is an order of magnitude scalable compared to straightforward multiplexing techniques such as servicing one client per disk head and round robin servicing of clients.

References

1. C. Abbott. Efficient Editing of Digital Sound on Disk. *Journal of Audio Engineering*, 32(6):394–402, June 1984.
2. J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
3. D. Ferrari and D. C. Verma. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.
4. J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 1992.
5. S. Gibbs, D. Tsichritzis, A. Fitas, D. Konstantas, and Y. Yeorgaroudakis. Muse: A Multi-Media Filing System. *IEEE Software*, 4(2):4–15, March 1987.
6. B.C. Ooi, A.D. Narasimhalu, K.Y. Wang, and I.F. Chang. Design of a Multi-Media File Server using Optical Disks for Office Applications. *IEEE Computer Society Office Automation Symposium, Gaithersburg, MD*, pages 157–163, April 1987.
7. Srinivas Ramanathan and P. Venkat Rangan. Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks. *IEEE/ACM Transactions on Networking*, 1(2):246–260, April 1993.
8. P. Venkat Rangan. Video Conferencing, File Storage, and Management in Multimedia Computer Systems. *Computer Networks and ISDN Systems*, 25:901–919, March 1993.
9. P. Venkat Rangan, Thomas Kaeppner, and Harrick M. Vin. Techniques for Efficient Storage of Digital Video and Audio. In *Proceedings of 1992 Workshop on Multimedia Information Systems (MMIS'92), Tempe, Arizona*, pages 68–85, February 1992.
10. P. Venkat Rangan and Harrick M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the 13th Symposium on Operating Systems Principles (SOSP'91), Operating Systems Review, Vol. 25, No. 5*, pages 81–94, October 1991.
11. P. Venkat Rangan and Harrick M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *To appear in the IEEE Transactions on Knowledge and Data Engineering*, August 1993.
12. P. Venkat Rangan, Harrick M. Vin, and Srinivas Ramanathan. Designing an On-Demand Multimedia Service. *IEEE Communications Magazine*, 30(7):56–65, July 1992.
13. Harrick M. Vin and P. Venkat Rangan. Designing a Multi-User HDTV Storage Server. *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, January 1993.