Designing File Systems for Digital Video and Audio

P. Venkat Rangan and Harrick M. Vin

Multimedia Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

We address the unique requirements of a multimedia file system such as continuous storage and retrieval of media, maintenance of synchronization between multiple media streams, and efficient manipulation of huge media objects. We present a model that relates disk and device characteristics to the recording rate, and derive storage *granularity* and *scattering* parameters that guarantee continuous access. In order for the file system to support multiple concurrent requests, we develop *admission control algorithms* for determining whether a new request can be accepted without violating the real-time constraints of any of the requests.

We define a *strand* as an immutable sequence of continuously recorded media samples, and then present a *multimedia rope* abstraction which is a collection of individual media strands tied together by synchronization information. We devise operations for efficient manipulation of multi-stranded ropes, and develop an algorithm for maintaining the scattering parameter during editing so as to guarantee continuous playback of edited ropes.

We have implemented a prototype multimedia file system, which serves as a testbed for experimenting with policies and algorithms for multimedia storage. We present our initial experiences with using the file system.

This work was supported by the IBM Corporation, the NSF Research Initiation Award No. NCR-9009757, Xerox Corp., UVC Corp., Compaq Corp., Starlight Networks Inc., NCR Corp., and the University of California MICRO program.

1 Introduction

1.1 Motivation

Future advances in networking and storage [2,3] will make it feasible for distributed systems to support multimedia services such as video and audio mail, news distribution, advertisement, and entertainment [11]. In this paper, we develop mechanisms for multimedia file storage and access, thereby taking us a step closer to multimedia computer systems.

Digital video and audio differ fundamentally from text in three important ways with regard to their storage requirements:

- *Multiple data streams*:

  A multimedia object consists of three components: audio, video, and text. Generally, these three components of a multimedia object are separated at the input and they arrive at the file system as three different streams. Similarly during retrieval, these streams are routed to three different output devices. Storing these media together may entail additional processing for combining them during storage, and for separating them during retrieval. The complexity of such processing can be significant if different encodings are used for the three media. On the other hand, if the three media are stored separately, the file system must explicitly maintain temporal relationships among the media so as to ensure synchronization between them during retrieval.

- *Continuous recording and retrieval of data streams*:

  Recording and playback of motion video and audio are continuous operations. The file system must organize multimedia data on disk so as to guarantee that their storage and retrieval proceed at their respective real-time rates.

- *Large file size*:

  Video and audio data have very large storage space requirements. If the file system is to act as a basis for supporting media services such as document editing, mail, distribution of news and entertainment, etc., it must provide mechanisms for manipulating and sharing stored data. For these mechanisms to be efficient on large sizes of multimedia data, they must minimize copying of data on the disk.

The design and implementation of a file system that addresses the above requirements of multimedia data is the subject matter of this paper.

1.2 Relation to Previous Work

Most of the multimedia file systems that are being built or proposed have focused on storage of still images and/or audio. The Diamond system [13], the Muse system of Gibbs et al [4], and the optical disk-based system of Ooi et al [10] are targeted towards storage and exchange of documents containing images. The Sun Multimedia File System [8], which consists of a collection of library functions built on top of Unix, is a storage scheme in which audio samples are stored as Unix files and shared among workstations using Sun's Network File System (NFS). The VOX audio server [1] also supports audio storage. Work by Mackay and Davenport [7] supports video filing, but video is stored in an analog form on consumer electronic devices. There has not been much work on storage systems for digital motion video. The Cambridge Pandora project [5] and Matsushita's Real Time Storage System [9] have begun investigating low level storage mechanisms for digital video.

Terry and Swinehart of the Etherphone project [12] present a powerful voice file system, in which sequences of intervals of voice samples form a *voice rope*. They present mechanisms for copy-free manipulation of voice, and a sophisticated reference count mechanism called *interests* for performing garbage collection of unreferenced voice. Our experience with the Etherphone system has been the initial motivation for the work reported in this paper.

1.3 Research contributions of this paper

Digitization of motion video yields a sequence of frames, and that of audio yields a sequence of samples. We call a sequence of continuously recorded video frames or audio samples a *Strand*. Each strand is organized on the disk in terms of blocks. The questions that we first attempt to answer in this paper are (1) how many video frames and/or audio samples are stored in each block (i.e., the storage *granularity*), and (2) how are the blocks constituting a media strand separated on the disk (i.e., the *scattering parameter*). In order to answer these questions, we relate disk and device characteristics (such as, disk read/write latency, and video capture/display times) to the recording rate, and derive the storage granularity and scattering parameter that result in continuous retrieval.

To enable the file system to support multiple concurrent storage/retrieval requests, we develop an *admission control algorithm* for determining whether a new request can be accepted without violating the real-time constraints of any of the requests.

A file system must not only store video and audio data, but also preserve temporal relationships among them. We present an abstraction called a *Multimedia Rope*, which is a collection of individual media strands tied together by synchronization information. (The term *rope* is derived from the Etherphone project). We devise operations for efficient manipulation of multi-stranded ropes, and develop an algorithm for maintaining the scattering parameter (so as to guarantee continuity of playback) without significant copying of data during insertion and deletion operations.

Using the above results, we have implemented a file system on an environment of SPARCstations and PC-ATs equipped with video compression hardware. We present our intial experiences with using the file system.

The rest of the paper is organized as follows: Section 2 defines the terminology used in the paper. In Section 3, we present methods to determine the granularity and scattering parameter of media storage. In Section 4, we define the structure of a multimedia rope, and discuss operations for its manipulation. Section 5 describes the software architecture of the multimedia file system that we have implemented. Finally, Section 6 summarizes the results and presents directions for future work.

2 Preliminary Definitions and Terminology

Frame is the basic unit of video.

Sample is the basic unit of audio.

Strand is an immutable sequence of continuously recorded audio samples or video frames. Immutability of strands is necessary to simplify the process of garbage collection.

Block is the basic unit of disk storage. There are two types of blocks: (1) **Homogenous blocks**, which contain data belonging to one medium, and (2) **Heterogenous blocks**, which contain data belonging to multiple media.

Rope is a collection of multiple strands (of same or different medium) tied together by synchronization information.

Table 1 defines the symbols used in this paper. Using these symbols, it can be seen that the duration of playback of a video block (which is the same as its recording duration) is given by $\frac{\eta_{vs}}{\mathcal{R}_{vr}}$, the total delay to read a video block from disk is given by $l_{ds} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}$, and the time to display a video block, which consists of the time for decompression and digital-to-analog conversion, is given by $\frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{vd}}$. Note that the time to display a block must not exceed the duration of its playback.

3 Determining Granularity and Scattering of Media Strands

A file system must divide video and audio strands into blocks while storing them on a disk. Most existing storage server architectures employ random allocation 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 access and latency times of successive blocks of the strand. Buffering can nullify the effects of unconstrained variation (i.e., jitter) in separations between blocks. The average seek time per block can be constrained by retrieving blocks in the order in which they are encountered while spanning all the cylinders of a disk (as opposed to the order in which they are to be played back), and then buffering the blocks until their playback. However, the number of blocks that need to be retrieved out of order and buffered can be as much as $\frac{l_{ds}^{adj} * n_{cyl}}{l_{ds}^{desired}}$, where, l_{ds}^{adj} is the maximum possible seek time between blocks on two adjacent cylinders on the disk, n_{cyl} is the total number of cylinders on the disk, and $l_{ds}^{desired}$ is the desired average seek time. Constrained block allocation, on the other hand, can yield the desired average seek time while minimizing the memory buffer requirements on media devices.

Partitioning a disk for multimedia and employing contiguous allocation of blocks within the partition can guarantee continuous access to blocks of a media strand, but it is fraught with inherent problems of fragmentation and can entail enormous copying overheads during insertions and deletions. Even the projected speeds of future fast disk configurations are not sufficient to ensure that unconstrained separation between blocks (i.e., the maximum possible access and latency times) lie within the requirements of high performance video applications. For example, with a block size of 4 Kbytes, future disk arrays with 100 parallel heads and projected seek and latency times of the order of 10 ms will be able to support 0.32 Gigabits/s transfer rates in the absence of constrained block allocation. This is inadequate for the retrieval of even one HDTV-quality video strand which may require data transfer rates of up to 2.5 Gigabit/s. Hence, constrained block allocation for storing media strands 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. A common file server can, however, integrate the functions of both a conventional text file server and a multimedia file server by employing constrained block allocation for (real-time) media strands, and using the gaps between successive blocks of a media strand to store text files.

There are two questions that need to be answered in constrained allocation of blocks of a media strand: (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 parameter) of a strand be? The guiding factor in determining the block size and separation is the requirement of continuous recording and retrieval. In the remainder of this section, we present a model that attempts to answer these questions and obtain the associated buffering requirements for motion video which is the most demanding medium (with regard to performance); the analysis for audio can be carried out in a similar manner.

We make two simplifying assumptions, both of which are reasonable in our hardware environment: (1) the disk write and read times are approximately equal, and (2) the time to capture a video frame (which consists of digitization and compression) and the time to display it (which consists of decompression and digital-to-analog conversion) are approximately equal. Hence, the continuity requirements of retrieval and storage are similar to each other, and in the analysis that follows, we only consider continuous retrieval.

3.1 Continuity Requirement

For continuous retrieval of media data, it is essential that media information be available at the display device at or before the time of its playback. We refer to this as the *'continuity requirement'*. Whereas the duration of playback of a media block stored on the disk is determined by the rate of recording; the time to access a disk block depends on the level of concurrency between disk access and video display. Disk access and video display can be purely sequential, or can be pipelined. Furthermore, if disks with multiple heads are used (such as RAIDs), multiple disk accesses can take place concurrently. We now analyze the sequential, pipelined, and concurrent architectures for continuity requirements.

Symbol	*Explaination*	*Unit*
$\mathcal{R}_{ar}$	Audio recording rate	samples/sec
$\mathcal{R}_{vr}$	Video recording rate	frames/sec
$\mathcal{R}_{dr}$	Rate of data transfer from disk	bits/sec
$\mathcal{R}_{vd}$	Rate of video display	bits/sec
η_{vs}	Granularity of video storage	frames/block
η_{as}	Granularity of audio storage	samples/block
s_{vf}	Size of a video frame	bits/frame
s_{as}	Size of audio sample	bits/sample
l_{ds}	Scattering parameter	sec

Table 1: Symbols used in this paper

Sequential architectures serialize read and display (similarly, capture and store) operations (see Figure 1). Each block is transferred from disk to a buffer in the video device, and then displayed before initiating the transfer of the next block.

The continuity requirement is met in this case if the sum of the time to read a block from disk and the time to display it does not exceed the duration of its playback. That is,

$$(l_{ds} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}) + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{vd}} \leq \frac{\eta_{vs}}{\mathcal{R}_{vr}} \quad (1)$$

Pipelined architectures perform read and display operations in parallel (see Figure 2).

If there are a minimum of two buffers on the video device, one holding the block being transferred and the other holding the block being displayed, the continuity requirement is met if the time to read a block does not exceed the duration of its playback. That is,

$$l_{ds} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}} \leq \frac{\eta_{vs}}{\mathcal{R}_{vr}} \quad (2)$$

Concurrent architectures perform multiple disk read operations in parallel. Let p be the degree of concurrency, i.e., the number of concurrent disk accesses.

If there are p buffers in the video device to hold the p blocks being transferred simultaneously (see 3), continuity of playback will be maintained if the time to read a block does not exceed the duration for playback of $(p-1)$ blocks. Hence,

$$l_{ds} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}} \leq (p-1) * \frac{\eta_{vs}}{\mathcal{R}_{vr}} \quad (3)$$

3.2 Synchronous Playback Requirement

In addition to maintaining continuity of retrieval, it is essential that the playback of a strand proceed at exactly the same rate as it was recorded. This is referred to as synchronous playback, and can be accomplished by one of two possible techniques:

- *Forced synchronization*: Using a clocking device, the display process can be forced to wait for appropriate time before displaying each block. This scheme entails communication overhead between clocking and display devices, and can be performed at the frame or block boundaries.
- *Automatic synchronization*: The left hand sides of Equations (1), (2), and (3) represent the effective access time per media block. If this becomes equal to the playback duration of a block, synchronization becomes automatic.

3.3 Discussion

3.3.1 Strict and Average Continuity Requirements

Recall that for continuous retrieval of media data, it is essential that media data be available at the display device at or before the time of its playback. Satisfying this condition deterministically for each block is referred to as *'strict continuity requirement'*, and is difficult to achieve in the presence of scheduling and seek time variations. By introducing anti-jitter delay at the beginning of each request, we can relax the continuity requirements so as to satisfy it on an average. Anti-jitter delay can be introduced by performing read-ahead of media blocks.

3.3.2 Buffering and Read-Ahead Requirements

When strict continuity requirements are satisfied, assuming buffers to be of the same size as disk blocks, the

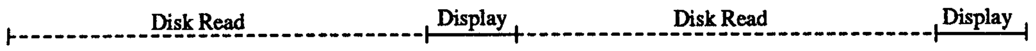


Figure 1: Sequential retrieval

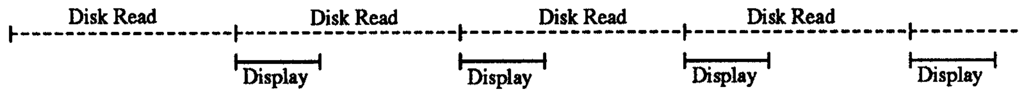


Figure 2: Pipelined retrieval

sequential, pipelined, and concurrent architectures require 1, 2, and p buffers, respectively. When continuity requirements are satisfied over an average of k successive blocks of a strand, in order to guarantee that the next group of k blocks can be retrieved from the disk within the time required to display a previous group of k blocks of a strand, the sequential and pipelined architectures require a read-ahead of k blocks, whereas the concurrent architecture requires a read-ahead of pk blocks (k for each of the p heads). In the case of sequential and concurrent transfers, the number of buffers required is same as the amount of read-ahead (i.e., k and pk, respectively), whereas, in the case of pipelined transfer, the number of buffers required is twice that amount, $2k$: one set of k buffers to hold the blocks being displayed, and another set of k buffers to hold the blocks being transferred from the disk, both of which occur simultaneously.

Functions such as fast-forwarding can be supported by satisfying continuity requirements at the fastest required display rate. Whereas fast-forwarding without skipping frames increases both continuity and buffering requirements, fast-forwarding with skipping increases only the continuity requirement. However, when blocks are displayed slower than the fastest rate (e.g., in slow motion), continuity requirements become over-satisfied, and retrieval of media blocks proceeds faster than their display, leading to accumulation of media blocks in buffers. In order to prevent unbounded accumulation, the disk can switch to some other task after all the buffers allocated to the retrieval of a media strand are filled, and switch back when sufficient buffers become empty. In order to compute the buffering needs in such a situation, note that after the disk switches to some other task, the disk head may have moved to a random location, and hence may have to incur maximum seek (and latency) time, l_{seek}^{max} before being able to resume the transfer of blocks of the earlier media strand. Thus, in order to guarantee that the display does not run out of media blocks during a switch to another task, the disk must read ahead an additional h blocks before the switch, given by,

$$h = l_{seek}^{max} * \frac{\mathcal{R}_{vr}}{\eta_{vs}}$$

where, $\frac{\mathcal{R}_{vr}}{\eta_{vs}}$ is the rate at which blocks are played back. Thus, the number of buffers would also be increased by h.

3.3.3 Storing Multiple Media Strands

The analysis presented so far has considered only one medium. There are two approaches for storing multiple media on a disk.

- *Heterogeneous Blocks*: Multiple media being recorded are stored within the same block, which may entail additional processing for combining these media during storage, and for separating them during retrieval. The advantage of this scheme is that it provides implicit inter-media synchronization.

- *Homogeneous Blocks*: Each block contains exactly one medium. This scheme permits the file system to exploit the properties of each medium to independently optimize its storage. However, the file system must maintain explicit temporal relationships among the media so as to ensure synchronization between them during retrieval.

We illustrate the analysis for deriving continuity equations for the pipelined architecture when there is one audio and one video component in the media source. For homogeneous blocks, the number of blocks to be retrieved increases with the number of media. Hence, if the duration of playback of audio block is n times that of a video block, an audio block is retrieved from disk for every n video blocks. Hence, the continuity requirement becomes

$$\frac{1}{n}(\underbrace{l_{ds}^{a} + \frac{\eta_{as} * s_{as}}{\mathcal{R}_{dr}}}_{Audio} + \underbrace{\sum_{i=1}^{n} l_{ds}^{v} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}}_{Video}) \leq \frac{\eta_{vs}}{\mathcal{R}_{vr}} \qquad (4)$$

On the other hand, if the duration of audio blocks is identical to that of video blocks (i.e., $n = 1$), then the continuity requirement reduces to

$$\underbrace{l_{ds}^{a} + \frac{\eta_{as} * s_{as}}{\mathcal{R}_{dr}}}_{Audio} + \underbrace{l_{ds}^{v} + \frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}}}_{Video} \leq \frac{\eta_{vs}}{\mathcal{R}_{vr}} \qquad (5)$$

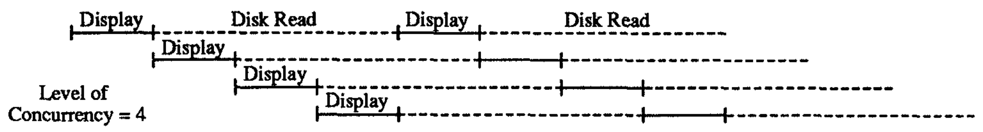


Figure 3: Concurrent retrieval

If the audio and video blocks are scattered on the disk such that $l^a_{ds} = 0$, then the continuity requirement reduces to that of the heterogeneous block case:

$$l^v_{ds} + \underbrace{\frac{\eta_{vs} * s_{vf}}{\mathcal{R}_{dr}} + \frac{\eta_{as} * s_{as}}{\mathcal{R}_{dr}}}_{Video+Audio} \leq \frac{\eta_{vs}}{\mathcal{R}_{vr}} \qquad (6)$$

3.3.4 Determining Granularity and Scattering

Having derived the continuity equations relating granularity of storage (η_{vs}) with the scattering parameter (l_{ds}), we discuss the process of determining each of these parameters for a given target environment.

Media blocks may be transferred from disk to a display device either *directly* to the internal buffers of the display device, or *through memory* (from disk to main memory, and then from main memory to the internal buffers of the display device). Whereas direct transfer is usually preferable, transfer through memory is more suitable for heterogeneous blocks. However, transfer through memory requires double the internal bus bandwidth. Hence, we will only consider the direct transfer approach below.

When direct transfer is used, the sizes of internal buffers available on the display devices can be used to determine the granularity of storage. For instance, if the video display device contains an internal buffer of the size of one video frame, the size of disk block can match this size, yielding $\eta_{vs} = 1$. On the other hand, if the internal buffers can store multiple frames (say f), then pipelined retrieval can be used by dividing the buffer into two parts, each of size $f/2$, and η_{vs} can be chosen anywhere in the range $1, ..., f/2$. If the disk permits p concurrent accesses, and the size of internal buffers is f frames, then η_{vs} can be chosen anywhere in the range $1, ..., f/p$.

Having determined the value of granularity η_{vs}, the upper bound of the scattering parameter l_{ds} can be obtained by direct substitution in the continuity equations.

3.4 Servicing Multiple Requests

In practice, a file server has to process requests from several clients simultaneously. Given a maximum rate of disk data transfer, the file system can only accept a limited number of requests without violating the continuity requirements of any of the requests. In this section, we formulate this resource allocation problem, and present an *admission control algorithm* for determining whether to accept a new request, given an existing set of requests being serviced.

Consider a scenario in which a file server is servicing n active media storage/retrieval requests. In order to service multiple requests simultaneously, the file system proceeds in rounds. In each round, it multiplexes among the media block transfers of the n requests. Let k_i, for $i \in [1, n]$, be the number of consecutive blocks retrieved for ith request before switching to the next request. Let $\eta^1_{vs}, \eta^2_{vs}, ..., \eta^n_{vs}$, and $\mathcal{R}^1_{vr}, \mathcal{R}^2_{vr}, ..., \mathcal{R}^n_{vr}$ be the granularities, and recording rates, respectively, of the strands corresponding to the n requests.

When the file server switches from one request to another, it may entail an overhead of up to the maximum disk seek time to move from a block in the first strand to a block of the second strand (since there is no guarantee on the relative positions of two strands belonging to two requests). The total time spent servicing ith request in each round can be divided into two parts:

1. θ^1_i: The overhead of switching from the previous request to the ith request, and then transferring the first block of ith request.

$$\Rightarrow \theta^1_i = l^{max}_{seek} + \frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}} \qquad (7)$$

2. θ^2_i: The time to transfer remaining $(k_i - 1)$ blocks of this request in this round.

$$\Rightarrow \theta^2_i = \sum_{j=1}^{k_i} (l^{ij}_{ds} + \frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}}) \qquad (8)$$

Hence, the total time spent servicing ith request in a round is

$$\theta_i = \theta^1_i + \theta^2_i \qquad (9)$$

The total time spent servicing one round of all the n requests is

$$\Theta = \sum_{i=1}^{n} \theta_i = n * l^{max}_{seek} + \sum_{i=1}^{n} (\frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}}) + \sum_{i=1}^{n} \sum_{j=1}^{k_i - 1} (l^{ij}_{ds} + \frac{\eta^i_{vs} * s^i_{vf}}{\mathcal{R}_{dr}}) \qquad (10)$$

The continuity requirement for each of the requests can be satisfied if and only if the service time per round does not exceed the minimum of the playback durations of all the requests. That is,

$$n * l_{seek}^{max} + \sum_{i=1}^{n}(\frac{\eta_{vs}^{i} * s_{vf}^{i}}{\mathcal{R}_{dr}}) +$$

$$\sum_{i=1}^{n}\sum_{j=1}^{k_i-1}(l_{ds}^{ij} + \frac{\eta_{vs}^{i} * s_{vf}^{i}}{\mathcal{R}_{dr}}) \leq \min_{i\in[1,n]}(k_i * \frac{\eta_{vs}^{i}}{\mathcal{R}_{vr}^{i}}) \qquad (11)$$

Thus, the file system can service all the n requests simultaneously if and only if $k_1, k_2, ..., k_n$ can be determined such that Equation (11) is satisfied. Determination of $k_1, k_2, ..., k_n$ in this most general formulation is beyond the scope of this paper. We make the following simplifying assumptions and develop an algorithm for admission control:

- The values of all k_i's are identical. That is, $k_1 = k_2 = ... = k_n = k$.
- We assume that

  $$\sum_{i=1}^{n}(\frac{\eta_{vs}^{i} * s_{vf}^{i}}{\mathcal{R}_{dr}}) \approx n * (\frac{\eta_{vs}^{avg} * s_{vf}^{avg}}{\mathcal{R}_{dr}})$$

  and

  $$\sum_{i=1}^{n}\sum_{j=1}^{k_i-1}(l_{ds}^{ij} + \frac{\eta_{vs}^{i} * s_{vf}^{i}}{\mathcal{R}_{dr}}) \approx n*(k-1)*(l_{ds}^{avg} + \frac{\eta_{vs}^{avg} * s_{vf}^{avg}}{\mathcal{R}_{dr}})$$

  where, individual values of the granularity, η_{vs}, the size of a video frame, s_{vf}, and the scattering parameter, l_{ds} are replaced by their respective averages in the summation.

We define

$$\alpha = l_{ds}^{max} + \frac{\eta_{vs}^{avg} * s_{vf}^{avg}}{\mathcal{R}_{dr}} \qquad (12)$$

$$\beta = l_{ds}^{avg} + \frac{\eta_{vs}^{avg} * s_{vf}^{avg}}{\mathcal{R}_{dr}} \qquad (13)$$

$$\gamma = \min_{i\in[1,n]} \frac{\eta_{vs}^{i}}{\mathcal{R}_{vr}^{i}} \qquad (14)$$

where, for a block of average granularity, α defines the maximum scattering, and β defines the average scattering. Note that since $l_{ds}^{max} \geq l_{ds}^{avg}$, it is guaranteed that $\alpha \geq \beta$. Under these assumptions, the continuity requirement of Equation (11) reduces to

$$n * \alpha + n * (k-1) * \beta \leq k * \gamma \qquad (15)$$

$$\Rightarrow \begin{array}{lcll} k & \geq & \frac{n(\alpha-\beta)}{(\gamma-n\beta)} & \text{if } \gamma > n\beta \\ k & \leq & \frac{n(\alpha-\beta)}{(n\beta-\gamma)} & \text{if } \gamma < n\beta \\ k & = & \infty & \text{if } \gamma = n\beta \end{array} \qquad (16)$$

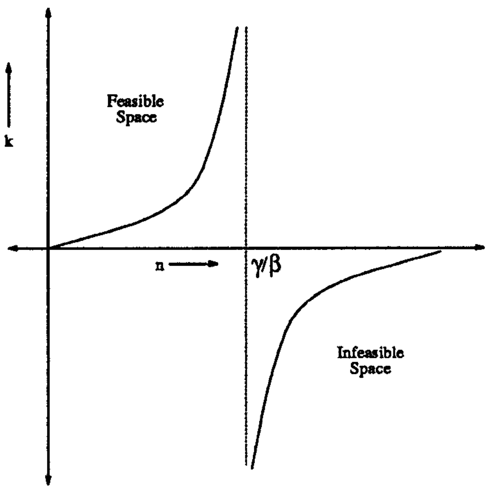


Figure 4: Variation of the number of blocks (k) with respect to the number of requests (n)

Figure 4 shows the variation of k with respect to n. Since k must be non-negative, the value of k obtained from Equation (16) is meaningful if and only if $\gamma > n\beta$. Thus, the maximum number of simultaneous requests that a file system can service is

$$n_{max} = \lceil \frac{\gamma}{\beta} - 1 \rceil \qquad (17)$$

It should be noted that, since the right hand side of Equation (15) represents the playback duration of the request with the fastest display rate, transferring k blocks of all other requests at that rate may lead to accumulation of data in the display subsystems of these other requests. Such an accumulation can be eliminated by regulating the number of data blocks transferred for each request during each service round, so as not to overflow the buffering available in the display subsystem of that request. Furthermore, larger the value of k, larger is the startup time for a new request. Thus, it is desirable to use the minimum possible value of k.

While servicing n requests, if the file server receives $(n+1)$th request, it must now decide whether or not to admit the new request or not. If $n+1 \leq n_{max}$ derived from Equation (17), it can determine the new values of α, β, and γ, and compute k_{new} (from Equation (16)) necessary for satisfying $(n+1)$ requests.

If $k_{new} = k_{old}$ (where, k_{old} is the value of k when the file system was servicing n requests), then it can immediately admit the $(n+1)$th request. However, if $k_{new} \neq k_{old}$, then $k_{new} > k_{old}$ (see Figure 4), and the file system has to begin transferring k_{new} blocks of each of the earlier n requests, and of the new $(n+1)$th request. During this round, the number of blocks being transferred is k_{new}, whereas, the number of blocks available for display are those of the previous round, which is k_{old}. Since, $k_{new} > k_{old}$, the time spent to transfer k_{new}

blocks may exceed the playback duration of k_{old} blocks of some of the requests, and a discontinuity may result in their playback. In other words, Equation (15) guarantees continuity only in steady state, and not during transitions.

In order to guarantee a smooth and transparent transition, we propose the following modification to Equation (15). Suppose the file system makes a transition from k_{old} to k_{new} in steps of 1 before beginning to service the $(n+1)$th request. When it performs a transition from k_{old} to $(k_{old}+1)$, the time to transfer $(k_{old}+1)$ blocks must not exceed the minimum playback duration of k_{old} blocks. Thus, if we use the time to transfer $(k+1)$ blocks instead of k in the left hand side of Equation (15) but use k in the right hand side, and then solve for k, a transparent transition from k_{old} to $(k_{old}+1)$ is guaranteed. Thus, Equation (15) changes to

$$n*\alpha + n*k*\beta \leq k*\gamma \tag{18}$$

Furthermore, since $\gamma \geq n\beta$,

$$n\alpha + nk\beta \leq k\gamma \Rightarrow n\alpha + n(k+1)\beta \leq (k+1)\gamma$$

Hence, a transition from $k_{old}+1$ to $k_{old}+2$, $k_{old}+2$ to $k_{old}+3$, ..., $k_{new}-1$ to k_{new} are also guaranteed. Thus, using Equation (18) to determine k, and increasing it in steps of 1, yields an admission control algorithm that guarantees *both* transient and steady state continuity.

3.5 Layout of Blocks in a Strand

A media strand consists of a sequence of *Media Blocks (MB)*, whose size and separation are determined using the techniques described in the previous sections. Each media block contains either video frames, audio samples, or both. A *3-level* index structure permits large strand sizes, and random as well as concurrent access to strands.

For each strand, the file system maintains primary indices in a sequence of *Primary Blocks (PB)*, each of which contains mapping from media block numbers to their raw disk addresses. Secondary indices, which are pointers to Primary Blocks, are maintained in a sequence of *Secondary Blocks (SB)* . Pointers to all Secondary Blocks of a strand are stored in the *Header Block (HB)*. A pictorial representation and the exact data structures of these blocks are shown in Figures 5 and 6, respectively.

4 From Media Strands to Multimedia Ropes

Multimedia data includes information in various forms: audio, video, textual, olfactory, thermal, tactile, etc.

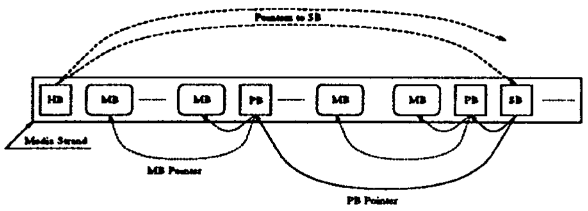


Figure 5: Organization of blocks constituting a media strand on disk

Primary Block [	
sector,	*– position of MB on disk*
sectorCount	*– length of the MB in sectors*
]	
Secondary Block [	
startBlock,	*– start Block number*
BlockCount,	*– number of Blocks in PB*
sector,	*– position of PB on disk*
sectorCount	*– length of PB in sectors*
]	
Header Block [	
frameRate,	*– Rate of recording*
secondaryCount,	*– Number of secondary blocks*
frameCount,	*– Total number of frames*
secondaryArray	*– Array of pointers to SB*
]	

Figure 6: Structure of 3-level indices of a media strand

All the media strands constituting a piece of information are tied together by inter-media synchronization to form a multimedia rope (see Figure 7). A rope contains the name of its creator, its length, access rights, and for each of its component media strands, the strand's unique ID (a NULL ID indicates the absence of that media in the rope), rate of recording, granularity of storage, and block-level correspondence (see Figure 8). The block-level correspondence information is used to synchronize the start of playback of all the media at strand interval boundaries. Within each strand interval, playing back at the strand's recording rate automatically guarantees simultaneity of playback between the media. Thus, the block-level correspondence and the recording rate information together maintain inter-media synchronization in multimedia ropes.

Maintenance of media synchronization information is complicated by silence detection and elimination of audio data. In silence elimination, if the average energy level over a block falls below a threshold, no audio data

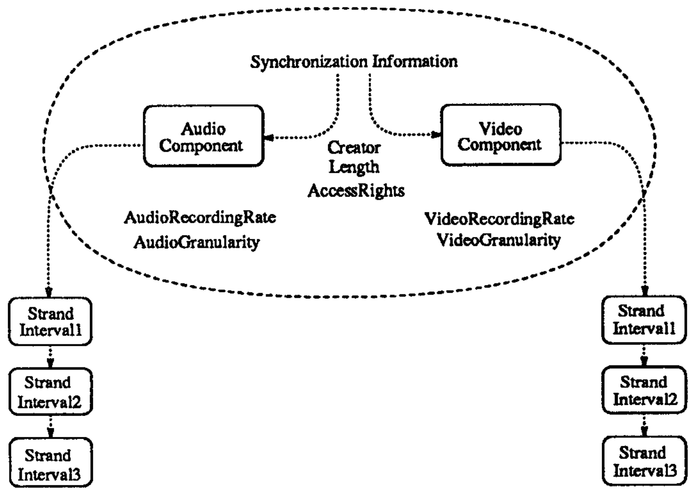


Figure 7: A multimedia rope containing video and audio

is stored for that duration. However, after undergoing silence elimination, audio strands no longer have lengths proportional to their duration. Hence, explicit delay holders have to be placed in audio strands to represent silences. We use NULL pointers in the primary blocks of a strand to indicate silence for the duration of a block.

In order to guarantee continuous retrieval, editing operations on ropes such as insert and delete may require substantial copying of their component strands. The strands can be very large in size and hence copying can consume significant amount of time and space. In order to minimize the amount of copying involved in editing, the multimedia file system regards strands as immutable objects, and all editing operations on ropes manipulate pointers to strands. Thus, an edited rope contains a list of pointers to intervals of strands. Many different ropes may share intervals of the same media strand. A media strand, no part of which is referred to by any rope, can be deleted to reclaim its storage space. A garbage collection algorithm such as the one presented by Terry and Swinehart in the Etherphone system [12], which uses a reference count mechanism called *interests*, can be used for this purpose. To simplify the process of garbage collection of media strands and ropes, synchronization information (which is typically very small in size) is copied from a rope to another when they share strands.

4.1 Operations on Multimedia Ropes

The file system provides facilities for creating, editing, and retrieving multimedia ropes. The exact interfaces are as follows:

RECORD [media] → [requestID, mmRopeID]

Assuming the user has the required access permissions, the file system begins recording a new multimedia rope (represented by mmRopeID) consisting of new media (audio, video or both) strands. RECORD invokes the continuity criteria obtained in the previous section to allocate free blocks for storing the media strands. If the media being recorded includes audio, then the file system performs silence detection and elimination. Recording continues until a subsequent STOP operation is issued.

PLAY [mmRopeID, interval, media] → requestID

Using the above interface, a user can retrieve any interval of any of the media of a previously recorded multimedia rope.

STOP [requestID]

When a user issues a STOP on an earlier PLAY or RECORD request, the retrieval or storage of the corresponding multimedia rope is halted.

Both RECORD and PLAY operations are non-blocking. Hence, a user may send multiple requests to the file system. The file system assigns a unique requestID to each request, and the clients use it to refer to the request subsequently. The file system accepts RECORD or PLAY requests using the admission control algorithm described in Section 3.4. Since RECORD and PLAY are continuous operations, it is desirable to allow a user to PAUSE (and later RESUME) a RECORD or a PLAY request, the file system provides the user the flexibility to specify either a *destructive* PAUSE, which causes resources to be deallocated during the PAUSE, or a *non-destructive* PAUSE, in which resources remain allocated. If a destructive PAUSE is specified, a subsequent RESUME will cause the file system to perform admission control.

In addition to RECORD, PLAY, PAUSE and RESUME operations, the file system also supports the following utilities:

INSERT[baseRope, position, media, withRope, withInterval]
REPLACE[baseRope, media, baseInterval,withRope, withInterval]
SUBSTRING[baseRope, media, interval]
CONCATE[mmRopeID1, mmRopeID2]
DELETE[baseRope, media, interval]

The functionality of these operations are similar to those on voice ropes in the Etherphone system. Figure 9 illustrates the withInterval of media strands of withRope, $Rope_2$ being INSERTed at position of baseRope, $Rope_1$. To guarantee real-time performance and continuity in retrieval operation, a small amount of copying of a strand may be necessary. We shall present an algorithm to bound this copying in the next section.

Any of the editing operations may be performed on any subset of media constituting a rope. An interesting

```
MultimediaRopeID                   - Unique ID
Creator                            - Identification of the creator
Length                             - Length of the rope in seconds
PlayAccess                         - List of user or group identifications
EditAccess                         - List of user or group identifications
List of [
    VideoStrand                    - Unique ID of video strand
    AudioStrand                    - Unique ID of audio strand
    Length                         - Length of the Strands in seconds
    VideoRecordingRate             - Rate of recording i.t.o frames/sec
    AudioRecordingRate             - Rate of recording audio
    VideoGranularity               - Granularity of video i.t.o. frames/block
    AudioGranularity               - Granularity of audio
    [AudioBlockID, VideoBlockID]   - Correspondence information
    List of [                      - Trigger information
        VideoBlockID               - Block number from video strand
        AudioBlockID               - Block number from audio strand
        TextString                 - Text to be synchronized with audio/video
        ]
    ]
```

Figure 8: Data structure representing a multimedia rope

application is to merge video and audio strands recorded separately to form a multimedia rope. For example, if *Rope*$_4$ contains only an audio strand, and *Rope*$_5$ contains only a video strand, then the operation

```
REPLACE[baseRope: Rope4, media: video,
        baseInterval: [start:0, length: L3],
        withRope: Rope5, withInterval:
                        [start:0, length:L3]]
```

replaces the non-existent video component of *Rope*$_4$ with the video component of *Rope*$_5$. The synchronization information for the resulting rope is generated by creating a correspondence between the blocks of the two strands.

4.2 Maintenance of Scattering while Editing

Editing operations such as insertion and deletion may cause a multimedia rope to consist of a sequence of intervals of media strands. While immutability of a media strand guarantees that their scattering parameter is bounded and hence the continuity requirement is satisfied within each of its intervals, the scattering parameter may not be bounded while moving from the last block of one interval (of a strand) to the first block of the next interval (which may belong to the same or another strand). Thus, discontinuities may be felt at interval boundaries during retrievals. These discontinuities can be eliminated by copying a small number of blocks of strands to which the intervals belong. We now present an algorithm to bound the number of blocks that need to be copied to guarantee continuity of retrieval.

Let us suppose that the result of an editing operation is a rope, one of whose components consists of intervals $[a_f, a_l]$ of strand S_a, and interval $[b_f, b_l]$ of strand S_b. Let the maximum possible separation between two blocks on a disk be l_{seek}^{max}. Suppose that the scattering parameters of strands S_a and S_b are not only bounded above (from continuity requirement) but also bounded below. Let the lower bounds on the scattering parameters of S_a and S_b be $l_{dsLower}^{a}$ and $l_{dsLower}^{b}$ respectively. Similarly, let the upper bounds on the scattering parameters be $l_{dsUpper}^{a}$ and $l_{dsUpper}^{b}$ respectively. Let

$$m = \frac{l_{seek}^{max}}{l_{dsLower}^{b}}$$

Note that the maximum separation between the last block of S_a (which is a_l) and block b_{f+m} of S_b is given by

$$\Delta_1 = l_{seek}^{max}$$

Also, the minimum separation between blocks $b_{f+m/2}$ and b_{f+m} of S_b is

$$\Delta_2 = \frac{m}{2} * l_{dsLower}^{b} = \frac{l_{seek}^{max}}{2}$$

Rope$_1$: < ..., VideoStrand: VS_1, AudioStrand: AS_1, Length: L_1, VideoRecordingRate: VR_1, AudioRecordingRate: AR_1, VideoGranularity: VG_1, AudioGranularity: AG_1, [VideoBlockID: VB_1, AudioBlockID: AB_1] >

Rope$_2$: < ..., VideoStrand: VS_2, AudioStrand: AS_2, Length: L_2, VideoRecordingRate: VR_2, AudioRecordingRate: AR_2, VideoGranularity: VG_2, AudioGranularity: AG_2, [VideoBlockID: VB_2, AudioBlockID: AB_2] >

$\Downarrow$

INSERT[baseRope: *Rope*$_1$, position: 100, media: AudioVisual, withRope: *Rope*$_2$, withInterval: [from: 0, length: 200]]

$\Downarrow$

Rope$_1$: < ...,
[..., VideoStrand: VS_1, AudioStrand: AS_1, Length: [start: 0, Length: 100], VideoRecordingRate: VR_1, AudioRecordingRate: AR_1, VideoGranularity: VG_1, AudioGranularity: AG_1, [VideoBlockID: VB_1, AudioBlockID: AB_1]],

[..., VideoStrand: VS_2, AudioStrand: AS_2, Length: [start: 0, Length: 200], VideoRecordingRate: VR_2, AudioRecordingRate: AR_2, VideoGranularity: VG_2, AudioGranularity: AG_2, [VideoBlockID: VB_2, AudioBlockID: AB_2]],

[..., VideoStrand: VS_1, AudioStrand: AS_1, Length: [start: 100, Length: $(L_1 - 100)$], VideoRecordingRate: VR_1, AudioRecordingRate: AR_1, VideoGranularity: VG_1, AudioGranularity: AG_1, [VideoBlockID: VB_1, AudioBlockID: AB_1]]
>

Figure 9: INSERT operation

Hence, the maximum separation between block a_l of S_a and block $b_{f+m/2}$ of S_b is given by

$$\Delta = \Delta_1 - \Delta_2 = \frac{l_{seek}^{max}}{2} = \frac{m}{2} * l_{dsLower}^{b}$$

in the best case (when the disk is sparsely occupied), and by

$$\Delta_1 = l_{seek}^{max} = m * l_{dsLower}^{b}$$

in the worst case (when the disk is densely occupied). Thus, when the disk is sparsely occupied, by redistributing $b_f, b_{f+1}, b_{f+2}, ..., b_{f+m/2-1}$ blocks equally in the region between block a_l of S_a and block $b_{f+m/2}$ of S_b, we can guarantee that the separation between a_l and b_f satisfies the bounds on the scattering parameter. Similar results hold when the disk is densely occupied, with block $b_{f+m/2-1}$ replaced by block b_{f+m-1}. Thus, the maximum number of blocks of S_b required to be copied is given by

$$C_b = \frac{m}{2} = \lceil \frac{l_{seek}^{max}}{2 * l_{dsLower}^{b}} \rceil \tag{19}$$

when the disk is sparsely occupied, which degrades to:

$$C_b = m = \lceil \frac{l_{seek}^{max}}{l_{dsLower}^{b}} \rceil \tag{20}$$

when the disk is densely occupied (i.e., nearly full).

Alternatively, instead of the first C_b blocks of S_b, we can redistribute the last C_a blocks of S_a, where C_a is computed in a similar manner. In practice, the actual number of blocks that needs to be copied is the minimum of C_a and C_b.

It should be noted that copying creates a new strand containing only the copied blocks because (1) strands are immutable, and (2) creating a separate strand aids the process of garbage collection. A unique ID is associated with this newly generated strand, and is used in the description of the multimedia rope created as a result of the editing operation.

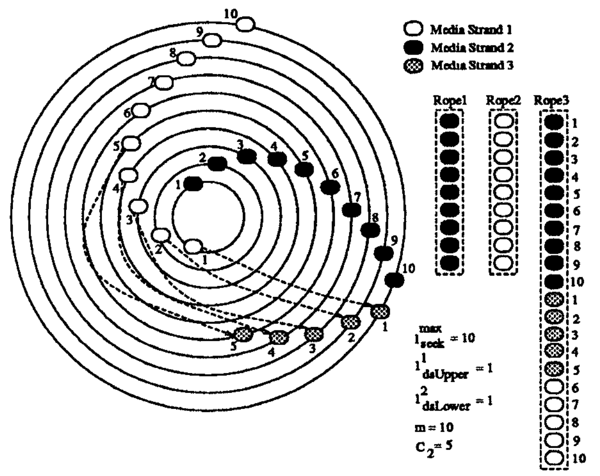


Figure 10: Copying of strands in editing operations (Rope3 ← CONCATE[Rope1, Rope2])

5 Experience with Multimedia File Storage and Management

We have implemented a prototype testbed multimedia file system to serve as a vehicle for experimenting with policies and algorithms outlined in this paper. The hardware environment and the software architecture of our testbed system are described in the following sections:

5.1 Hardware Environment

Our Multimedia Laboratory is equipped with a number of multimedia stations, each consisting of a Sun SPARCstation, a PC-AT, a video camera, and a TV monitor (see Figure 11). The SPARCstations and PC-ATs are connected via Ethernets. The PC-ATs are equipped with digital video and audio processing hardware produced by UVC Corporation [6]. The audio hardware digitizes audio signals at 8 KBytes/sec. The video hardware can digitize and compress motion video at real-time rate up to NTSC broadcast with a resolution of 480x200 pixels and 12 bits of color information per pixel. Video data is stored on the local disk attached to the PC-AT, and displayed on a monitor attached to it. The operation of the PC-ATs is controlled via SPARCstations. Communication between the SPARCstations and PCs is accomplished using TCP/IP socket library.

Note that the separation of the media-processing functionality from the workstation provides reliability, performance, and flexibility: audio and video processing peripherals provide reliable media processing without compromising workstation performance on other tasks, and users of different workstations can use the media processing features without making any modifications to their workstation hardware.

5.2 Software Architecture

The software architecture of the prototype file system was designed to serve as a testbed for experimenting with various policies described in this paper. There are two main functional layers: the *Multimedia Storage Manager* (MSM) and the *Multimedia Rope Server* (MRS).

- *Multimedia storage manager*: This layer is responsible for physical storage of media strands on the disk. The functionality of the MSM include: determination of granularity and scattering of strands, enforcing admission control to service multiple requests simultaneously, and maintenance of scattering while editing.
- *Multimedia Rope Server*: This layer is responsible for creating and maintaining the multimedia ropes. It supports all the rope manipulation operations.

The rationale for the above layering is that:

- A decoupled design of the MRS and the MSM permits their execution on different hardware. In addition, it facilitates easy experimentation with various policies in one layer without effecting the other.
- The MRS implements the device-independent multimedia rope abstraction. The MSM implements storage device-specific algorithms, and hence, is hardware dependent.

The MRS of our testbed system is implemented on a SPARCstation, whereas the MSM is implemented on a PC-AT. Applications are compiled with a rope stub library which uses remote procedure calls to contact the MRS. The first application we implemented that uses the file system is a window-based editor to manipulate multimedia ropes. Figure 12 shows a typical editing session with the editor.

6 Concluding Remarks

6.1 Summary

We have analyzed the unique requirements of a multimedia file system such as continuous storage and retrieval of media, maintenance of synchronization between multiple media streams, and efficient manipulation of huge media objects. We have presented a model that relates disk and device characteristics (such as, disk read/write latency, and video capture/display times) to the recording rate, and derived storage granularity and scattering parameters that guarantee continuous access. The continuity requirements define an upper bound on the scattering parameter. The algorithm that bounds

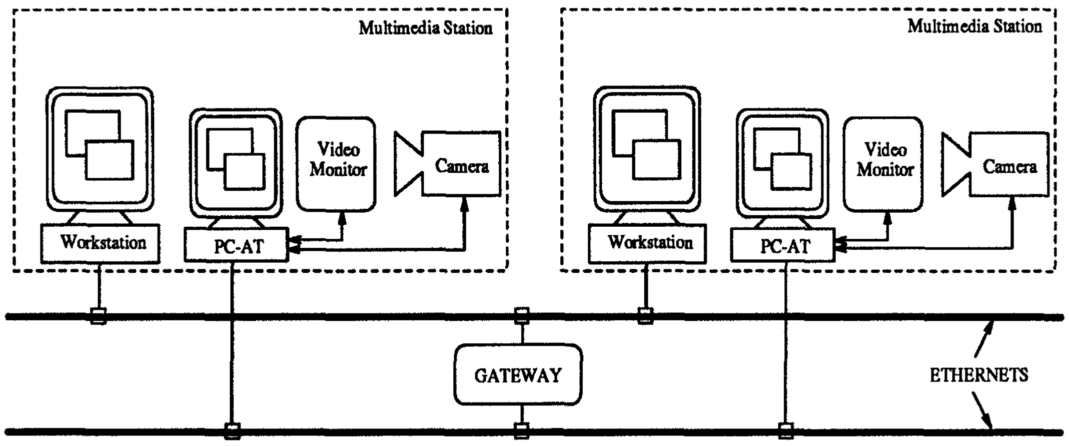


Figure 11: System configuration

the amount of copying necessary during editing operations define the lower bound of the scattering parameter. Thus, the separation between consecutive blocks of a strand must be chosen within these bounds.

In order to support multiple concurrent requests, we have presented an admission control algorithm that determines whether a new request can be accepted without violating the real-time constraints of any of the requests. The algorithm guarantees both transient and steady state continuity.

We have defined strand and rope abstractions, and have outlined an approach to maintain synchronization information among strands. We have described editing operations for multi-stranded ropes.

6.2 Future Work

In the storage model presented in this paper, we have assumed that video frames and disk blocks are of fixed size. However, variable rate compression of video (analogous to silence elimination in audio), such as differencing between frames, can result in varying but smaller sizes of video frames, thereby yielding better bounds for granularity and scattering. We are extending the continuity equations to incorporate such effects of compression algorithms.

Constrained scattering of blocks of a media strand can be difficult to achieve when the disk is densely utilized. When it becomes impossible to place new media strands in such a way that their scattering bounds are satisfied, the storage of existing media strands on the disk may have to be reorganized. Towards this end, we are investigating mechanisms for merging multiple media strands so as to optimize storage utilization, and we are studying techniques by which a small number of anamolies in scattering can be smoothed out.

The admission control algorithm that we have developed uses a round-robin servicing of requests in the order in which they are received, and assumes maximum separation between blocks while switching between requests. As a result, the estimates of the maximum number of requests that can be simultaneously serviced are pessimistic. We are investigating algorithms for servicing requests in the order that minimizes (possibly, in a statistical sense) the separations between blocks, thereby minimizing the overhead of switching between requests, and optimizing the maximum number of requests that can be serviced simultaneously.

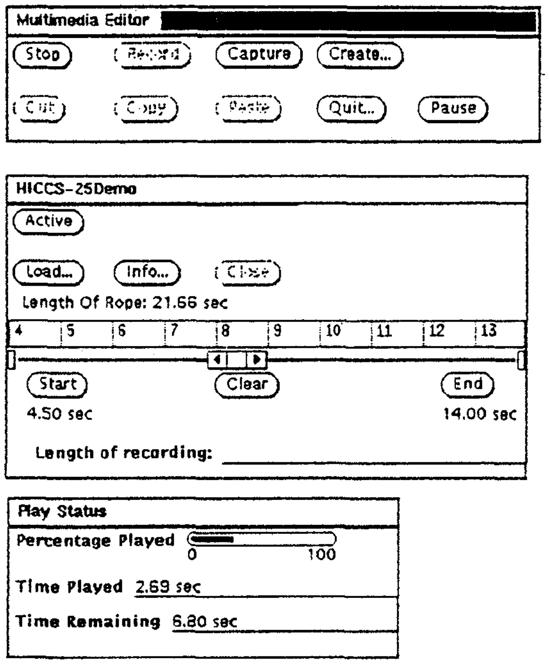
We have implemented a prototype multimedia file system, which serves as a testbed for experimentation. We are enhancing the prototype to (1) permit access over a network, and (2) provide conversational interface so that it can be accessed from within multimedia conferences.

Acknowledgement

The initial motivation for this work comes from our involvement in the Etherphone project, which was made possible by Dan Swinehart, Doug Terry, and Polle Zelleweger. We are thankful to Walt Burkhard, Robert Bowdidge, Kashun Chan, Ingvar Aaberg, John Lindwall, Ljubisa Radivojevic, Linda Yamamoto, and Ian Harris for their contributions towards implementing the prototype multimedia file system, to Srinivas Ramanathan for helping in preparing the final version of the paper, and to Bill Walton of Compaq, Thomas Kaeppner, David Reed, and anonymous referees for their insightful comments on the paper.

References

[1] S. Angebranndt, R.L. Hyde, D.H. Luong, N. Siravara, and C.Schmandt. Integrating Audio and



Length Of Rope: 21.66 sec

4.50 sec

14.00 sec

Length of recording:

Percentage Played

Time Played 2.69 sec

Time Remaining 6.80 sec

Figure 12: Window-based Multimedia Editor

Telephony in a Distributed Workstation Environment. In *Proceedings of Summer 1991 Usenix Conference*, pages 419–436, June 1991.

[2] P. Cochrane and M. Brain. Future Optical Fiber Transmission Tech. and Networks. *IEEE Communications Magazine*, 26(11):45–60, November 1988.

[3] J. Gait. The Optical File Cabinet: A Random-Access File System for Write-Once Optical Disks. *IEEE Computer*, 21(6):11–22, June 1988.

[4] 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.

[5] A. Hopper. Pandora – an experimental system for multimedia applications. *ACM Operating Systems Review*, 24(2):19–34, April 1990.

[6] M. Leonard. Compression Chip Handles Real-Time Video and Audio. *Electronic Design*, 38(23):43–48, December 1990.

[7] W.E. Mackay and G. Davenport. Virtual Video Editing in Interactive Multimedia Applications. *Communications of the ACM*, 32(7):802–810, July 1989.

[8] Sun Microsystems. Multimedia File System. *Software Release*, August 1989.

[9] Y. Mori. Multimedia Real-Time File System. In *Matshushita Electric Industrial Co.*, February 1990. private communication.

[10] 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.

[11] C.S. Skrzypczak. The Intelligent Home of 2010. *IEEE Communications Magazine*, 25(12):81–84, December 1987.

[12] D.B. Terry and D.C. Swinehart. Managing Stored Voice in the Etherphone System. *ACM Transactions on Computer Systems*, pages 3–27, February 1988.

[13] R.H. Thomas, H.C. Forsdick, T.R. Crowley, R.W. Schaaf, R.S. Tomlinsin, V.M. Travers, and G.G. Robertson. Diamond: A Multimedia Message System Built on a Distributed Architecture. *IEEE Computer*, 18(12):65–78, Dec. 1985.