To appear in the *IEEE Transactions on Knowledge and Data Engineering*

Special Issue on Multimedia Information Systems, August 1993

Efficient Storage Techniques for Digital Continuous Multimedia

P. Venkat Rangan & *Harrick M. Vin*

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, Fax: (619) 534-7029

Abstract

Advances in storage and networking have led to the development of multimedia computing systems capable of offering services such as multimedia mail, news distribution, advertisement, and entertainment. Supporting such services requires the integration of storage and transmission of multimedia data with computing. In this paper, we address the problem of collocational storage of media strands, which are sequences of continuously recorded audio samples or video frames, on disk. We present a model that relates disk and device characteristics to the playback rates of media strands, and derives storage patterns so as to guarantee continuous retrieval of media strands. To efficiently utilize the disk space, we develop mechanisms for merging storage patterns of multiple media strands, by filling the gaps between media blocks of one strand with media blocks of other strands. We propose both an on-line algorithm suitable for merging a new media strand into a set of already stored strands, and an off-line merging algorithm that can be applied a priori to the storage of a set of media strands before any of them have been stored on disk. As a consequence of merging, storage patterns of media strands may become perturbed slightly, to compensate which, read-ahead and buffering are required so that continuity of retrieval remains satisfied. We present techniques for minimizing both read-ahead and buffering. These techniques are being implemented in the multimedia storage server being developed at the UCSD Multimedia Laboratory.

Index terms: Digital video and audio storage, continuous retrieval, merging techniques, read ahead, buffering

للاستشارات المنارة

www.manaraa.com

1 Introduction

Future advances in networking will make it feasible for digital computer networks to support video transmission [3, 12]. Coupled with the rapid advances in storage technologies, they can be used to build services such as multimedia mail, news distribution, advertisement, and entertainment over metropolitan-area networks such as B-ISDN [2, 11]. Supporting such services requires the integration of storage and transmission of multimedia data with computing. Whereas transmission of digital multimedia data will have to wait a few more years for ultra-high bandwidth networks to become pervasive, the integration of multimedia storage with distributed computing merits immediate attention.

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

- *Multiplicity of media streams*: A multimedia object consists of several media components (such as audio and video), which, generally, are separated at the input and arrive at the storage server as different streams. Storing these media together may entail additional processing for combining them during storage, and for separating them during retrieval. Storing them separately, on the other hand, requires that the storage server maintain explicit temporal relationships among the media so as to ensure synchronous retrieval.

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

- *Large data size*: Video and audio data have very large storage space requirements. If the storage system is to act as a basis for supporting document editing, mail, distribution of news and entertainment, etc., it must organize multimedia data on the disk so as to efficiently use the limited available space.

The design of a high-performance multimedia storage server that addresses the above requirements of real-time digital video and audio is the subject matter of this paper. Specifically, we define a sequence of continuously recorded audio samples or video frames as a *Strand*, and address the problem of collocational storage of multiple media strands on disk. We present a model that relates disk and device characteristics to the playback rate of media strands, and for each media strand, derive a storage pattern consisting of a media block size and an inter-block gap size that can guarantee the strand's continuous retrieval. To efficiently utilize the disk space, we develop

المنارة للاستشارات
www.manaraa.com

mechanisms for merging storage patterns of multiple media strands by filling the gaps between media blocks of one strand with media blocks of other strands, but without violating the continuity requirements of either of the strands. We propose both an on-line algorithm suitable for merging a new media strand into a set of already stored strands, and an off-line merging algorithm that can be applied a priori to a set of media strands before any of them have been stored on disk. The off-line algorithm can operate with much smaller buffer sizes, and it does so by using a staggered toggling technique in which sizes of successive media blocks are fine tuned individually so as not to exceed the available buffer size. As a consequence of merging, the storage pattern, and hence the continuity requirements, of media strands may not be maintained strictly for each media block. However, it is possible to introduce read-ahead and buffering of a finite number of blocks so as to preserve the storage pattern and continuity properties at least on an average over a finite number of blocks. We present techniques for determining the amount of read-ahead and buffering required to guarantee continuous retrieval of merged media strands. The merging algorithms and techniques presented in this paper form a basis for a prototype multimedia storage server being implemented at the UCSD Multimedia Laboratory [11].

The rest of this paper is organized as follows: In Section 2, we present a model for deriving storage patterns of individual strands, and then review previous work in this area. In Section 3, we develop the algorithms for merging multiple strands. Techniques for determining read-ahead and buffering of merged media strands are presented in Section 4. In Section 5, we describe our prototype multimedia server and present its preliminary performance results, and finally, Section 6 concludes the paper.

2 A Framework for Efficient 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 storage server 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. Such storage servers cannot handle media strands 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. At the other end of the spectrum, contiguous allocation of blocks of a strand can guarantee continuous access, but it is fraught with inherent problems of fragmentation and can entail enormous copying overheads during insertions and deletions. Constrained block allocation, on the other hand, can keep the access time within media playback requirements without entailing the above disadvantages.

المنارة للاستشارات
www.manaraa.com

Symbol	Explanation	Unit
M	Media block size in the storage pattern of a strand	sectors
G	Gap size in the storage pattern of a strand	sectors
$\mathcal{B}$	Buffer size needed during the retrieval of a strand	bytes
$\mathcal{R}$	Read ahead necessary for retrieval of a strand	bytes
ρ	Rate of data transfer to or from disk	sectors/sec
δ	Display time of a media block	sec

Table 1: Symbols used in this paper

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 be? and (2) What should the separation between successive blocks of a strand be? The guiding factor in determining the block size and the separation is the requirement of continuous retrieval of media strands at their respective playback rates. Continuous retrieval of media strands can be guaranteed if the time to skip over a gap and retrieve the next media block does not exceed the duration of its playback. Using the symbols defined in Table 1, the condition for continuous retrieval can be formulated as:

$$\frac{M+G}{\rho} \leq \delta \tag{1}$$

For each strand, the relative values of its media block size M and its separation between successive media blocks G must satisfy Equation (1), which we refer to as the *continuity requirement*. Since there are two parameters and one equation, one of these parameters, namely the media block size, can be determined based on the hardware environment and the amount of buffer space available at the display devices. Having fixed M, the upper bound on G can be obtained by direct substitution in Equation (1). We refer to the pair (M, G) as the storage pattern of a strand[1]. For example, if a HDTV quality video strand is digitized at 0.5 Mbits/frame and recorded at 60 frames/s on a disk with data transfer rate of 25600 sectors/s (each sector equals 512 bytes, yielding a transfer rate of 100 Mbits/s), then choosing each media block to contain one video frame yields a storage pattern (M, G) to be (125, 425) sectors. A strand consists of repetitions of this pattern (see Figure 1).

Note that user interaction using functions such as fast-forwarding can be supported by satisfying continuity requirements at the fastest required display rate (i.e., at the smallest value of δ). However, when blocks are

[1]Clearly, any increase in the value of M (which effectively increases the read-ahead) is also accompanied by a corresponding increase in G. In the extreme case, if M is chosen such that G exceeds the maximum seek and rotational latencies, then constrained allocation of media blocks on disk degenerates to unconstrained allocation.

المنارة للاستشارات
www.manaraa.com

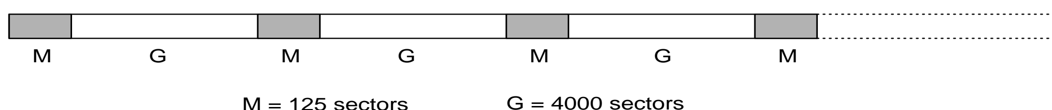

Figure 1: Representation of media strands as repetitions of storage patterns

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 [10].

Even though bounding the separation between successive media blocks so as not to exceed G ensures continuous retrieval of media strands, the separation between successive media blocks (namely, $\frac{G}{\rho}$) may, in general, be significantly larger than the time to read a media block from disk (namely, $\frac{M}{\rho}$). Hence, if a placement policy is based solely on G, 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 fact, if the separation between successive media blocks is set to be exactly equal to $(\delta * \rho - M)$, then a multimedia storage server can retrieve only one strand at a time. In order to support simultaneous retrieval of multiple strands, 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 storage server (such as, the delay incurred in initiating a new disk block access after having completed a previous request), and defines a lower bound G' on the separation between successive media blocks on disk. If a multimedia storage server ensures that the separation between each pair of successive media blocks is within $[G', G]$, then it is referred to as *strict* placement algorithm. On the contrary, an *adaptive* placement algorithm may accommodate occasional violations of the bounds on the separation between successive media blocks as long as the average separation over a finite window of blocks is within $[G', G]$. 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. Both of these placement algorithms can exploit varying playback durations of media blocks (yielded from storing variable number of compressed media units in each media block) by dynamically computing the separation between successive media blocks, and then appropriately adjusting the layout of media strands on disk.

المنارة للاستشارات
www.manaraa.com

In practice, a multimedia storage server needs to store thousands of media strands on disk. If there are sufficiently large empty regions on the disk, each strand may be stored exactly in accordance with its storage pattern. However, storing each strand independently entails the unusability of all the gaps in its storage pattern, resulting in an occupancy of $\frac{M}{M+G}$, which, for the values of M and G computed earlier, is about $\frac{1}{5}$. In order to *utilize the disk space efficiently*, blocks of a new strand may have to be stored in the gaps of already existing strands on the disk. We refer to this process as *merging*. Depending on whether or not they preserve the individual storage patterns of the media strands at the time of merging, merging policies can be classified into the following two categories:

- *Storage Pattern Preserving (SPP) policies*, which merge a set of media strands only if the storage patterns of each of the strands can be strictly preserved even after being merged. That is, even after merging, each media block of a merged strand must exactly equal M contiguous sectors, and the separation between every two successive media blocks must exactly equal G contiguous sectors, thereby guaranteeing continuous retrieval.

- *Storage Pattern Altering (SPA) policies*, on the other hand, may distribute media blocks of a strand being merged into the gaps of existing strands, even if that causes breaks in the storage pattern of the merged strand.

The SPP merging policies have been studied by Yu et al. [13]. But, because of their strict insistence on exactly preserving the storage pattern of each strand, the SPP merging policy turns out to be inflexible and space inefficient for multiple heterogeneous media strands.

SPA merging policies, since they cause perturbations in the storage patterns of media strands, cannot guarantee continuous retrieval of merged media strands. However, they maintain the relative ratio of the size of media blocks and gaps for each merged strand to be $\frac{M}{G}$, at least *on an average over a finite length of the strand*. Consequently, by introducing read-ahead and buffering of a finite number of media blocks of each strand, so as to nullify the effects of jitter due to perturbations in its storage pattern, continuous retrieval can be reinsured. In this paper, *we develop algorithms for SPA merging of multiple media strands*, and obtain the read-ahead and buffering requirements necessary for continuous retrieval.

In the recent past, multimedia storage systems have begun to receive much attention. However, most of the past work is restricted to still images and/or audio [1, 5, 8]. Work by Rangan and Swinehart [9] supports

المنارة للاستشارات
www.manaraa.com

video filing, but video is stored in an analog form on consumer electronic devices. The Matsushita's Real Time Storage System [7] has investigated some of the low level storage mechanisms for digital video. Gammell and Christodoulakis [4] have described file system designs for supporting multiple playback channels of delay sensitive data. However, their scheme assumes contiguous storage of media strands on disk. Whereas contiguous storage of media strands ensures efficient retrieval of individual strands, retrieval of a multimedia object (consisting of several media components - such as, audio and video), stored on disk as several different contiguous strands, may incur significant seek and rotational latencies due to switching between strands. In order to alleviate this shortcoming, a model for the design of a file system for digital video and audio based on constrained block allocation was first proposed by Rangan and Vin in [10]. In the next few sections, we significantly extend that model by developing mechanisms for merging the storage of multiple synchronous strands, thereby minimizing the overhead due to seek and the rotational latencies incurred during their concurrent retrieval.

3 Merging Techniques

Media strands may arrive for storage at a storage server either in an isolated fashion, one at a time, each belonging to a different multimedia object, or simultaneously, many of them at the same time, which will in fact be the case if all of them belong to the same multimedia object. In the former case, a newly arriving media strand should be merged into strands that already exist on the disk, whose laid out storage patterns cannot be changed, and is termed: *On-line Merging*. In the latter case, called *off-line merging*, the strand patterns of the simultaneously arriving multiple media strands can be mutually adjusted prior to merging, so as to minimize read-ahead and buffering requirements for continuous retrieval after their merged placement on disk. In the next two subsections, we develop algorithms for on-line and off-line merging, respectively.

3.1 On-line Merging

Consider two media strands S_1 and S_2 with storage patterns (M_1, G_1) and (M_2, G_2), respectively. Let the blocks of strand S_1 be laid out on the disk in accordance with its storage pattern, thereby guaranteeing continuous retrieval of S_1. To efficiently utilize the storage space, the file system must store the blocks of S_2 in the gaps of the pattern for S_1. The constraint in this process of merging is that the continuity requirement of S_2 must not be violated.

Merging of S_2 into S_1 is straight-forward if $G_1 = M_2$ and $M_1 = G_2$. In such a case, each media block of S_2 will exactly fit into a gap of S_1. However, in general, this can be very restrictive. We will now derive

المنارة للاستشارات
www.manaraa.com

the conditions for deciding whether the storage of two media strands can be merged together. Throughout this analysis, we will assume that the lengths of media strands may not be bounded at the time of the start of its recording, but that the read-ahead and the buffer sizes are required to be bounded.

Intuitively, a set of strands $S_1, S_2, ..., S_n$ can be merged together if the sum of the fractions of space occupied by media blocks of $S_1, S_2, ..., S_n$ does not exceed 1:

$$\frac{M_1}{M_1 + G_1} + \frac{M_2}{M_2 + G_2} + \cdots + \frac{M_n}{M_n + G_n} \leq 1 \qquad (2)$$

We analyze this condition first for two strands, and then extend the analysis to multiple strands.

3.1.1 Binary On-line Merging

If we simplify Equation (2) for two strands, we obtain:

$$\frac{M_1}{M_1 + G_1} + \frac{M_2}{M_2 + G_2} \leq 1$$

$$\Rightarrow \frac{M_1}{M_1 + G_1} \leq 1 - \frac{M_2}{M_2 + G_2} = \frac{G_2}{M_2 + G_2}$$

$$\Rightarrow G_1 * G_2 \geq M_1 * M_2$$

The following proposition formally shows that the above condition is both necessary and sufficient for merging of two strands:

Proposition 1 (Merge Condition) *Let S_1 and S_2 be two strands with associated storage patterns (M_1, G_1) and (M_2, G_2), respectively. We can merge strand S_2 into strand S_1 if and only if,*

$$G_1 * G_2 \geq M_1 * M_2 \qquad (3)$$

Proof: In order to merge strand S_2 into strand S_1, the media blocks in S_2's pattern over a finite length L must be accommodated in the gaps of S_1's pattern of the *same length*. This is because, since the storage pattern of S_2 has been determined from continuity requirements, it is guaranteed that the time to display all of S_2's media blocks over any finite length L is sufficient to scan and transfer L sectors from the disk. Thus, after merger, by reading ahead and buffering S_2's media blocks contained within a chunk of length L of S_1's pattern, S_2's continuity of display is guaranteed to be satisfied.

المنارة للاستشارات
www.manaraa.com

We will first show that, if Equation (3) is satisfied, then, over a length L equal to the lowest common multiple of the patterns of strands S_1 and S_2, all the media blocks of S_2 can be accommodated within the gaps of S_1, and hence S_2 can be merged into S_1. Let $L = \text{LCM}(M_1+G_1, M_2+G_2)$ be the lowest common multiple of the patterns of S_1 and S_2, from which we can obtain k_1 and k_2 such that $k_1*(M_1+G_1) = L$ and $k_2*(M_2+G_2) = L$ (see Figures 2(a) and 2(b)). That is, L is the number of sectors in which k_1 instances of S_1's pattern or k_2 instances of S_2's pattern fit. Given that,

$$G_1 * G_2 \geq M_1 * M_2$$

we obtain:

$$\Leftrightarrow G_1 * G_2 + G_1 * M_2 \geq M_1 * M_2 + G_1 * M_2$$

$$\Leftrightarrow G_1 * (G_2 + M_2) \geq M_2 * (M_1 + G_1)$$

$$\Leftrightarrow G_1 * k_1 * k_2 * (G_2 + M_2) \geq M_2 * k_1 * k_2 * (M_1 + G_1)$$

$$\Leftrightarrow G_1 * k_1 * L \geq M_2 * k_2 * L$$

$$\Leftrightarrow G_1 * k_1 \geq M_2 * k_2 \tag{4}$$

Thus, k_2 media blocks of strand S_2 fit in k_1 gaps of strand S_1, thereby showing that length L of S_2 can be accommodated in the same length of S_1.

On the other hand, suppose that it is possible to merge strand S_2 into strand S_1, that is (as explained at the beginning of this proof), there exists a finite length L such that a segment of length L of strand S_2 fits within a segment of the same length of strand S_1. We will now show that Equation (3) must hold.

Let us suppose that, k_1 patterns of S_1 or k_2 patterns of S_2 span a length L:

$$k_1 * (M_1 + G_1) = L \tag{5}$$

$$k_2 * (M_2 + G_2) = L \tag{6}$$

Since length L of S_2 fits into the same length of S_1, k_2 media blocks of S_2 must fit in k_1 gaps of S_1:

$$k_2 * M_2 \leq k_1 * G_1$$

Substituting for k_1 and k_2 from Equations (5) and (6) respectively, and simplifying yields that:

$$G_1 * G_2 \geq M_1 * M_2$$

المنارة للاستشارات
www.manaraa.com

which is nothing but Equation (3).

□

If the merge condition is satisfied, the simplest way to lay out S_2's media blocks is to fill them into S_1's gaps continuously starting from the very first gap. After storing S_2's media blocks in a chunk of length L of S_2's pattern in the first few gaps of a chunk of the same length L of S_1's pattern, the remaining gaps in that chunk of length L of S_1's pattern would be left free. In this layout policy, which we term as *greedy*, if S_2's pattern is sparse compared to the empty space available in S_1, a large number of media blocks of S_2 are read earlier than their time of display, leading to peaks in buffering requirements (see Figure 2(c)).

The buffering requirements can be reduced by distributing the media blocks of S_2 *uniformly* over all the gaps of S_1, so as to guarantee that the separation between every pair of consecutive media blocks of S_2 is almost the same (see Figure 2(d)). Formally, if k_1 patterns of strand S_1 and k_2 patterns of strand S_2 span over a merge cycle of length L, then we define the pseudo block size of S_2 with respect to S_1 as the smallest integer M_2' such that

$$k_2 * M_2' \geq k_1 * G_1 \tag{7}$$

Since the storage patterns of S_1 and S_2 satisfy the merge condition, we know that $k_2 * M_2 \leq k_1 * G_1$, yielding that $M_2' \geq M_2$. If $M_2' = M_2$, then the merge condition is exactly satisfied, and the media blocks of S_2 occupy all the gaps of S_1. If $M_2' > M_2$, then in each merge cycle, we can distribute the k_2 blocks of S_2 such that there are either $M_2' - M_2 - 1$ or $M_2' - M_2$ free sectors between consecutive media blocks of S_2. (To be precise, in each merge cycle, $k_2 * M_2' - k_1 * G_1$ blocks will have a separation of $M_2' - M_2 - 1$ sectors, and the remaining will have a separation of $M_2' - M_2$ sectors). The $(k_2 + 1)$th block of S_2 will be stored exactly L sectors away from the first media block of S_2, resulting in a pattern of filled and free sectors that repeats after every L sectors, which is the length of one *merge cycle* (see Figure 2). Such a distribution is almost uniform in the sense that the number of free sectors between successive blocks of S_2 can differ only by one.

3.1.2 On-line Merging of More than Two Strands

The above binary merging techniques can be easily extended to the storage of three or more strands. Let us suppose that we have to merge n strands, S_1, S_2, S_3, ..., S_n. When the first two strands S_1 and S_2 are merged, the resulting storage pattern can be viewed as that of a *composite strand* $S_{1,2}$, whose media block size $M_{1,2}$ and gap

المنارة للاستشارات
www.manaraa.com

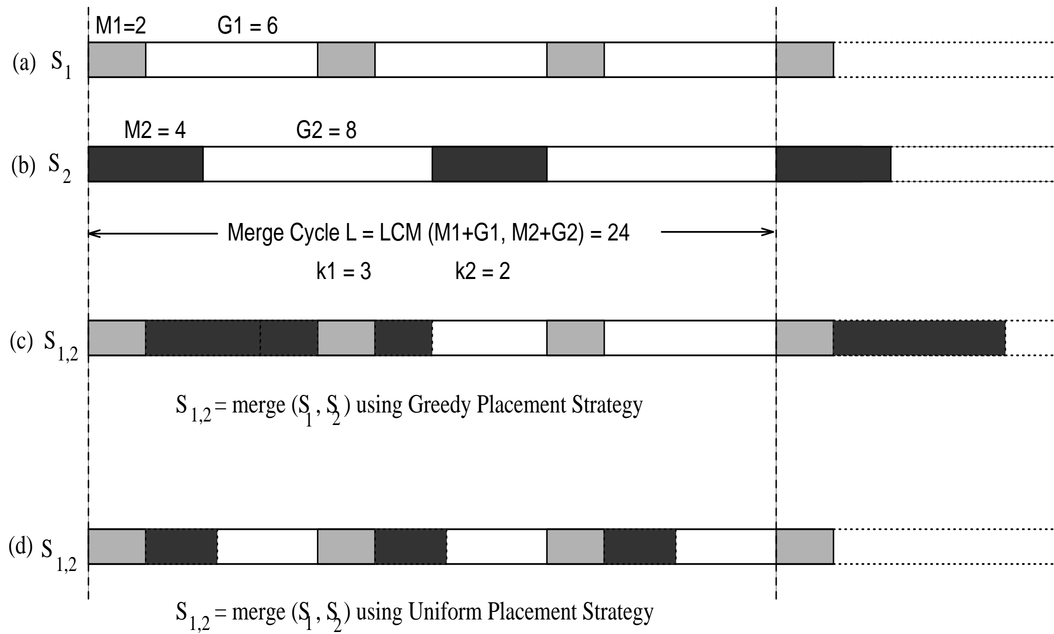

Figure 2: On-line merging of strand S_2 into S_1

size $G_{1,2}$ are given by:

$$M_{1,2} = k_1 * M_1 + k_2 * M_2$$

$$G_{1,2} = L_{1,2} - M_{1,2}$$

where $L_{1,2} = \text{LCM}(M_1 + G_1, M_2 + G_2)$ is the length of the pattern, $k_1 = \frac{L_{1,2}}{M_1+G_1}$, and $k_2 = \frac{L_{1,2}}{M_2+G_2}$. The merger of S_3 with $S_{1,2}$ can now be carried out using the binary merging techniques to yield strand $S_{1,2,3}$. Continuing the merge operation in this fashion, the n strands can be merged using a sequence of $n-1$ binary merge operations, resulting in a final composite strand, $S_{1,2,3,\ldots n}$

Deletion of a strand S_i from a composite strand $S_{1,2,\ldots,n}$ releases the sectors occupied by the media blocks of strand S_i to the pool of gaps of the composite strand $S_{1,2,\ldots,n}$. Let (M_i, G_i) and $(M_{1,2,\ldots,n}, G_{1,2,\ldots,n})$ denote the storage patterns of strands S_i and $S_{1,2,\ldots,n}$, respectively, and that k_i patterns of strand S_i exist in each pattern of $S_{1,2,\ldots,n}$. Then, deleting S_i from $S_{1,2,\ldots,n}$ yields a strand $S'_{1,2,\ldots,n}$ with storage pattern (M'_i, G'_i), where $M'_{1,2,\ldots,n} = M_{1,2,\ldots,n} - k_i * M_i$ and $G'_{1,2,\ldots,n} = G_{1,2,\ldots,n} + k_i * M_i$. Consequently, the length of the storage pattern of the composite strand, namely $(M'_{1,2,\ldots,n} + G'_{1,2,\ldots,n})$, remains the same. If, however, the strand to be deleted represents the last one to be merged, then deletion restores the pattern to that before the last merge, thereby

المنارة للاستشارات
www.manaraa.com

decreasing the pattern length of the composite strand. For example, if strand S_n is deleted from a composite strand $S_{1,2,\ldots,n}$, the parameters $L_{1,2,\ldots,n}$, $G_{1,2,\ldots,n}$, and $M_{1,2,\ldots,n}$ are restored back to $L_{1,2,\ldots,n-1}$, $G_{1,2,\ldots,n-1}$, and $M_{1,2,\ldots,n-1}$, respectively, which are the values prior to the merging of strand S_n into $S_{1,2,\ldots n-1}$.

Notice that, as more and more strands are merged together, gaps become more and more scarce. Hence, larger merge cycle lengths (L) are necessary for storing the media blocks of newer strands, leading to increases in buffer space requirements. On the contrary, if none of the strands that need to be stored by a multimedia storage server have been physically placed on the disk, the patterns of each of the strands can be determined so as to be exactly mergeable, thereby eliminating pattern deviations for any of the strands when they are stored in a merged form, and consequently reducing the buffer space requirements. Such an off-line merging technique, suitable for the placement of media strands on write-once optical disks (such as, WORMs and CLVs), is elaborated next.

3.2 Off-line Merging

Suppose that media strands $S_1, S_2, \ldots, S_n$ with storage patterns (M_1, G_1), (M_2, G_2), ..., (M_n, G_n), respectively, are to be stored in a merged form on the disk. In off-line merging, suppose that we place the strands on disk such that chunks of k_1 blocks of S_1, k_2 blocks of S_2, ..., k_n blocks of S_n follow each other, and the sequence repeats indefinitely (see Figure 3).

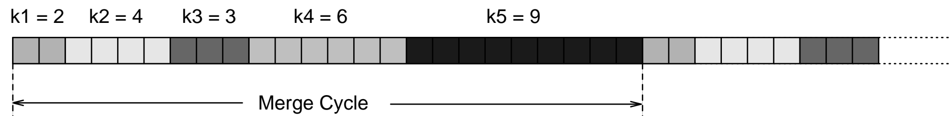

Figure 3: Off-line merging

Guaranteeing retrieval at its playback rate for each strand S_i requires that the space occupied by blocks of all 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 the k_i blocks (present in each chunk) of S_i. That is,

$$\forall \text{ strands } S_i, i \in [1, n] : \sum_{j \in [1,n], j \neq i} k_j * M_j \leq k_i * G_i \tag{8}$$

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 of strands placed in a merge cycle are inversely related to their pattern length.

المنارة للاستشارات
www.manaraa.com

3.2.1 Scaled Placement Policy for Off-line Merging

In the scaled placement policy, the number of consecutive blocks k_i of a strand S_i placed in a merge cycle is inversely scaled by its pattern length (i.e., $M_i + G_i$). That is, $\forall i \in [1, n]$:

$$k_i = \frac{k}{M_i + G_i} \tag{9}$$

where, k is a constant. The following theorem proves that the scaled placement policy will always yield a solution if one exists, thereby showing that it is complete in its effectiveness.

Theorem 1 *Whenever the merge condition (Equation (2)) is satisfied, the scaled placement policy always yields a merge cycle.*

Proof: In the scaled placement policy, the number of consecutive blocks k_i in a chunk of a strand S_i placed in a merge cycle is given by $k_i = \frac{k}{M_i+G_i}$. The set of values of k_i so fixed constitute a merge cycle if they satisfy Equation (8), which reduces to:

$$\forall j \in [1, n] : \sum_{i \in [1,n], i \neq j} \frac{k * M_i}{M_i + G_i} \leq \frac{k * G_j}{M_j + G_j}$$

$$\Rightarrow \forall j \in [1, n] : \sum_{i \in [1,n], i \neq j} \frac{M_i}{M_i + G_i} \leq \frac{G_j}{M_j + G_j} \tag{10}$$

Substituting $\frac{G_j}{M_j+G_j} = 1 - \frac{M_j}{M_j+G_j}$ in Equation (10), which is surprisingly independent of k, and rearranging terms, we get:

$$\sum_{i=1}^{n} \frac{M_i}{M_i + G_i} \leq 1$$

which is nothing but the merge condition (Equation (2)), which goes to prove that, the scaled placement policy yields a solution whenever the merge condition is satisfied.

□

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. At a display device, 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 Section 4, we will present techniques for computing the exact buffering requirements of a merged strand). In turn, given the bounds on buffering available at display

المنارة للاستشارات
www.manaraa.com

devices, 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 its value, from which the tightest values of k_i are recomputed, again by using Equation (9). The k_i's so computed are used in the scaled placement policy for off-line merging of media strands.

However, the values of k_i so obtained may not be integral (unless k is chosen to be an integral multiple of LCM$(M_1 + G_1, M_2 + G_2, ..., M_n + G_n)$, which can, of course, be prohibitively large). Since the display of media strands typically proceeds in terms of quanta such as frames, assuming each media block contains a display quantum (as in our hardware environment), retrieval of a fraction of a block placed in a merge cycle cannot be used for display. Consequently, the display will have to starve until the remaining fraction of the block arrives, which will be at the beginning of retrieval of the next merge cycle. Hence, it is essential that the values of $\{k_1, k_2, ..., k_n\}$ all be integers. Deriving integral values by uniformly truncating or rounding off k_i's obtained from the tightest value of k (which itself is derived from buffering limitations, as described earlier), may violate Equation (8) and thereby violate continuity constraints. We now describe a technique for toggling between $\lfloor k_i \rfloor$ and $\lceil k_i \rceil$ from one merge cycle to the next in a staggered manner among strands, so as to guarantee that continuity of retrieval of all the media strands remain satisfied without ever overflowing the available buffers at display devices.

3.2.2 Scaled Placement with Integral Quanta

Let the values of $\{k_1, k_2, ..., k_n\}$ yielded by Equation (9) (upon using the tightest value of k obtained from display buffering limitations) be:

$$\forall i \in [1, n] : \quad k_i = I_i + F_i$$

where I_i and F_i are the integer and the fractional parts, respectively, of k_i. If $I = \sum_{i=1}^{n} I_i$ and $F = \sum_{i=1}^{n} F_i$, then $(I + F)$ denotes the average number of media blocks that constitute a merge cycle. In the technique that we present, the number of blocks laid out for a strand S_i within a merge cycle toggles between $\lfloor k_i \rfloor$ and $\lceil k_i \rceil$, so that on an average, the number of blocks of strand S_i stored in a merge cycle is k_i. However, in doing so, continuous retrieval requirements that would have been guaranteed to have been met by $\{k_1, k_2, ..., k_n\}$, must continue to be satisfied. In particular, since a fractional block cannot be used for display, for any merge cycle D_i, the total number of blocks of each strand S_i, stored upto and including D_i, must equal at least the integral number of blocks that would have been available for display, had k_i blocks of S_i been stored in each merge cycle. Formally,

المنارة للاستشارات
www.manaraa.com

if $\mathcal{K}_c = \{k_1^c, k_2^c, \cdots k_n^c\}$, where k_i^c can equal either I_i or $(I_i + 1)$, is the set of numbers of blocks of the n strands stored in a merge cycle c, it should be the case that:

$$\forall i \in [1, n] : \sum_{c=1}^{D_i} k_i^c \geq \lfloor D_i * k_i \rfloor \tag{11}$$

In order to layout the media strand S_i so as to satisfy this inequality, $I_i = \lfloor k_i \rfloor$ blocks are stored in each merge cycle c, until a *deadline merge cycle* D_i, in which the difference $(D_i * k_i - \sum_{c=1}^{D_i} k_i^c)$ just reaches or goes past a value of 1 for the first time, reversing the inequality (11). Since I_i and k_i can differ by at most 1, the inequality can be re-established in the deadline merge cycle by storing $I_i + 1$ blocks instead of I_i.

The earliest deadline merge cycle can be computed from Equation (11), assuming I_i (and not $I_i + 1$) blocks have been stored in all preceding merge cycles, i.e., $\forall c \in [1, D_i] : k_i^c = I_i$, in which case, inequality (11) reduces to:

$$D_i * (k_i - I_i) \geq 1$$

yielding the equation for the earliest deadline merge cycle as:

$$D_i = \lceil \frac{1}{k_i - I_i} \rceil = \lceil \frac{1}{F_i} \rceil \tag{12}$$

Later deadline merge cycles can be computed iteratively in a similar manner.

Having computed the deadline merge cycles for each strand S_i, the storage server need not wait until such cycles to store $I_i + 1$ (instead of I_i) blocks of the strand. Alternatively, the server can order the strands in the order of earliest deadline merge cycle first, and as soon as it finds enough extra space in a merge cycle, store the additional blocks for strands in that order. This way, the toggling of I_i to $(I_i + 1)$ for strands are dynamically staggered so as never to exceed available space in each merge cycle. However, waiting until their respective deadline merge cycles to transfer additional blocks of strands has the advantage of not requiring any more buffers than that for which k_i's were derived[2] (earlier in Section 3.2.1).

It can be shown that, in the above technique, for each strand S_i, there will always be sufficient space to store its additional block at or before its deadline merge cycle. To see why, consider the accumulated *slack space* available in a deadline merge cycle:

$$Slack(D_i) = D_i * k_i - \sum_{c=1}^{D_i} k_i^c = D_i * (k_i - I_i)$$

[2]That the number of buffers is the same whether k_i's or k_i^c's are used, is evident from the following observation: When additional blocks are placed in deadline merge cycles, the LHS and RHS of Equation (11), which relates the k_i^c's to k_i's, can differ by at most a fraction; assuming fixed size buffers, such a fractional difference does not reduce the number of buffers.

www.manaraa.com

Substituting for D_i from Equation 12, we obtain the desired result:

$$Slack(D_i) \geq 1$$

showing that there will always be space for an additional block in a deadline merge cycle.

4 Determining the Read-Ahead and Buffering

During retrieval of a media strand, media blocks are both (1) continuously retrieved from the disk and added to an intermediate buffer by the multimedia server, and (2) continuously removed from the buffer by the display device. If the strand were to be stored in accordance with its unmerged pattern, and continuity requirements were satisfied exactly, each block would be transferred to the buffer just prior to its playback time. However, an important consequence of merging (either on-line or off-line) is that the continuity requirements, which hold for every block in a strand's pattern, are now guaranteed to hold only on an average over an entire merge cycle. Within a merge cycle, the media blocks of a strand may be scattered in the gaps of other strands that were available at the time of its merging. Thus, in order to guarantee timely availability of media blocks for continuous playback, a read ahead of at most one merge cycle may be necessary before initiating playback. The media blocks that are retrieved during a read ahead need to be stored in the intermediate buffer between the multimedia server and the display device. The buffer occupancy (i.e., the number of media blocks in buffer) starts with being equal to the read ahead, and grows and diminishes over a period of a merge cycle depending on the actual layout of media blocks. We now present techniques for computing the exact read-ahead and buffering requirements for continuous playback of merged strands.

Consider the retrieval of a merged strand from disk. At the time when the multimedia server is transferring blocks at a position x on the disk, let $M(x)$ denote the number of media blocks retrieved from the disk and $D(x)$ the number displayed. Hence, the buffer occupancy $B(x)$ is given by:

$$B(x) = M(x) - D(x)$$

Figure 4, referred to as the *buffer occupancy chart*, graphically illustrates the above equation, with $D(x)$ being linearly increasing because of an assumed constant display rate, and $M(x)$ consisting of a succession of ramps, with each ramp representing the retrieval of a media block, and the horizontal steps between the ramps representing gaps between media blocks.

www.manaraa.com

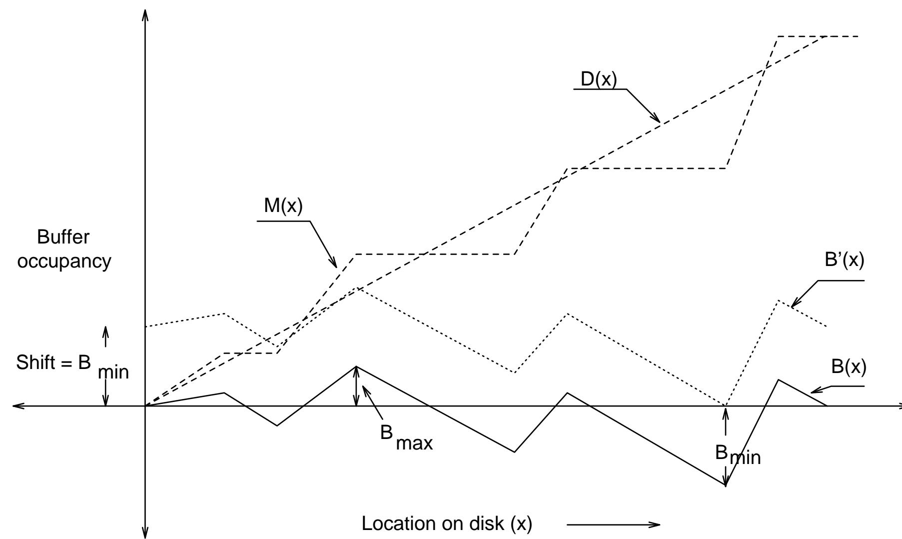

Figure 4: Buffer occupancy chart

If the buffer occupancy were to ever become negative, then, at that instant, the display device would be starved of the required media block, resulting in a loss of continuity. Such a discontinuity can only be averted by a read ahead: the amount of read ahead must be such that, in its presence, the buffer can be empty at minimum, i.e., the buffer occupancy curve must be shifted so as to touch the x-axis at its lowest point (such a shifted curve is represented by dotted curve in Figure 4). Specifically, if

$$B_{min} = |\min_{\forall x} B(x)|$$

and

$$B_{max} = \max_{\forall x} B(x)$$

then the dotted curve represents

$$B'(x) = B(x) + B_{min}$$

which is nothing but $B(x)$ shifted upwards by B_{min}.

The exact amount of read ahead that is necessary depends on where the retrieval starts on the disk[3], and can be as much as the difference in the height between the highest and lowest points on the buffer curve (which could

[3]The read ahead need not be B_{min}, except when the retrieval starts at some position to the left of the origin (assuming the media strand stretches to infinity on both sides of the origin), in such a way that at the end of the read ahead, the disk head is positioned at location zero (at which instant the display is also initiated).

المنارة للاستشارات
www.manaraa.com

be the case, for instance, if the retrieval were to start just before the location where the curve reaches the maximum height), given by: $B_{min} + B_{max}$.

We will now present a technique to precisely compute the read ahead, given that the retrieval starts at an arbitrary location X_{start} (see Figure 5). If the buffer occupancy were to follow the curve $B'(x)$, playback is guaranteed to be exactly continuous (since the buffer occupancy never becomes negative), in which case, when the disk head is retrieving the media block from location X_{start}, the buffer would contain $B'(X_{start}) = B(X_{start}) + B_{min}$ media blocks. Consequently, media blocks retrieved at location X_{start} would be displayed only after the time to display $B'(X_{start})$ media blocks, given by: $T_{init}(X_{start}) = B'(X_{start}) * \delta$, has elapsed. During this time, the disk head would have advanced by $B'(X_{start}) * \delta * \rho$ sectors past the location X_{start}, i.e., the disk head would be positioned at:

$$X_{init} = X_{start} + B'(X_{start}) * \delta * \rho$$

At the instant X_{start}'s media block is displayed, the disk head is positioned at X_{init}, and the buffer occupancy is $B'(X_{init})$. Since the curve $B'(x)$ represents a retrieval/display equilibrium process that exactly satisfies continuity requirements, $B'(X_{init})$ is also the smallest buffer occupancy that is needed at the time of display of X_{start}'s media block. Thus, if the media strand were to be retrieved and displayed starting from X_{start}, $B'(X_{init})$ would become the initial read ahead necessary, and is given by:

$$\mathcal{R}(X_{start}) = B'(X_{init}) = B(X_{start} + (B(X_{start}) + B_{min}) * \delta * \rho) + B_{min}$$

The time $T_{init}(X_{start}) = B'(X_{start}) * \delta$ represents the initiation latency from the time of retrieval start to the time of display start, during which period the read ahead is performed.

5 Experience and Performance Evaluation

At the UCSD Multimedia Laboratory, we are developing a prototype multimedia server on a 486-PC, having a disk configuration consisting of 10 platters, each with 823 tracks. Each track is arranged in 64 sectors of 512 bytes each, and hence, has a capacity of 32768 bytes/track. The maximum rotational latency of the disk is 16.66 ms, and the data transfer rate is: $\rho = 3840$ sectors/second. The multimedia server provides video and audio on-demand recording and retrieval services to several multimedia stations connected via ethernets and FDDI (see Figure 6). Each multimedia station consists of a Sun SPARCstation, a PC-AT, a video camera, and a display monitor. The

المنارة للاستشارات
www.manaraa.com

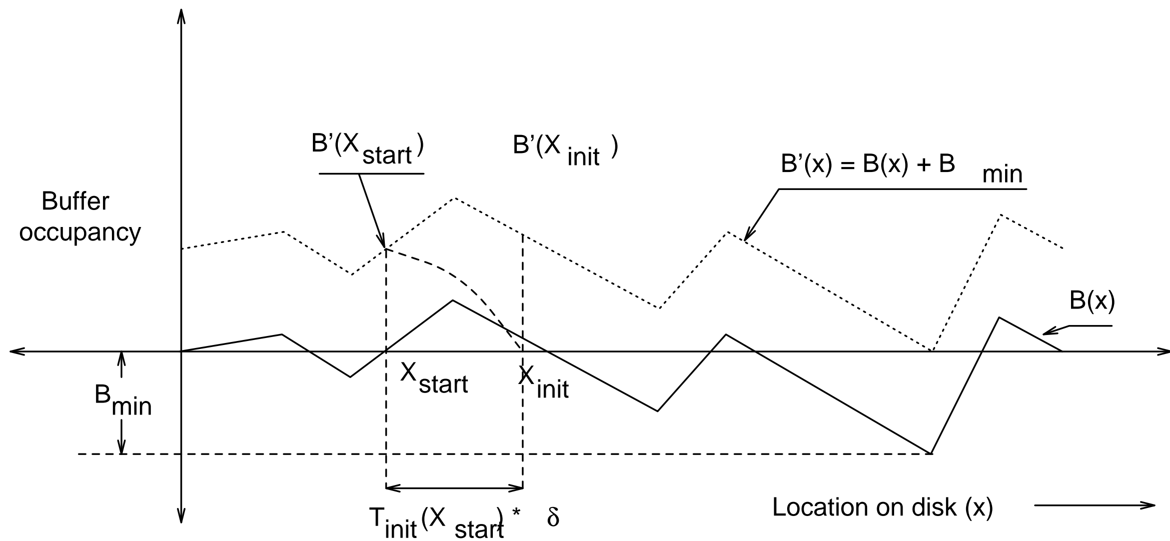

Figure 5: Determination of read-ahead and buffering requirements

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 rates.

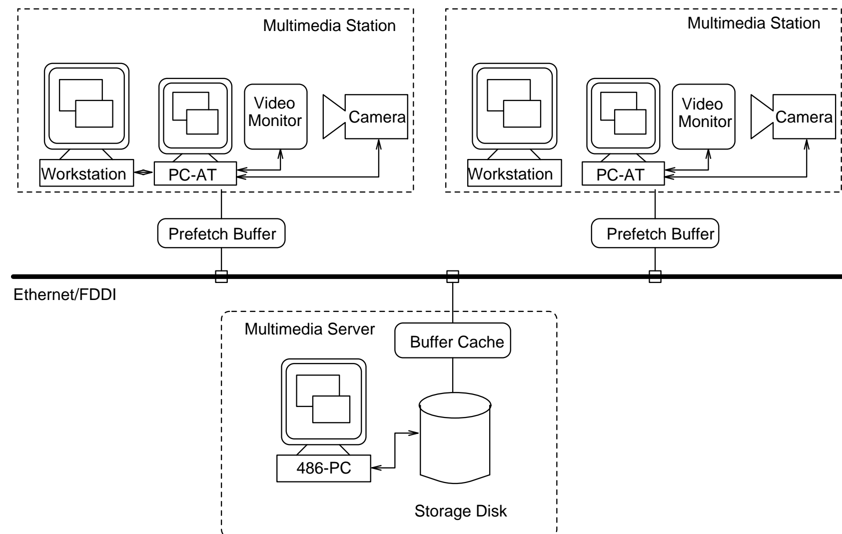

Figure 6: Hardware configuration of our multimedia server prototype

We have implemented the on-line and off-line merging algorithms and carried out their preliminary performance estimation for video strands (which are the most demanding in terms of performance) whose recording rates could be fixed at one of six levels: 32 or 16 frames/sec, 30 or 15 frames/sec, and 20 or 10 frames/sec (which

www.manaraa.com

are the operational frame rates of our hardware video digitizer). For each of these six strands, media blocks were chosen to be the smallest possible, i.e., containing one frame each, with each frame occupying about 25 sectors (which is about 12.8 Kbytes). The resulting gap sizes computed from Equation (1) are: 95, 103, 167, 215, 231, and 359 sectors, respectively.

Under these constraints, when the strands are stored independently (i.e., without merging), then the average storage efficiency is found to be about 14%. On the contrary, both on-line and off-line merging techniques yield a storage efficiency of about 80%, thereby providing evidence of the significant improvements in storage utilization that can be expected due to merging.

For the on-line algorithm, the read-ahead and buffering requirements increase with the increase in the fragmentation of a strands's blocks, and hence, yield highest values for the last merged strand. This is the case when the gap size at the time the last strand arrives is the minimum, requiring that the strands arrive in the order of smallest gap size first, which is nothing but highest recording rate first (since the media block size is assumed to be the same for all the strands). For such a scenario, Table 2 shows the large increases in merge cycle lengths accompanying each successive merger of the video strands using the on-line algorithm, with the cycle length reaching a maximum of 11520 sectors when the sixth and the final strand (with recording rate of 10 frames/sec) is merged.

Recording rates of sequences of merged strands	Merge cycle length (in sectors)
{32}	120
{32, 30}	1920
{32, 30, 20}	1920
{32, 30, 20, 16}	1920
{32, 30, 20, 16, 15}	3840
{32, 30, 20, 16, 15, 10}	11520

Table 2: Prohibitive increase in the merge cycle length in on-line merging

For the same above mentioned scenario, Figures 7 and 8 show the buffer utilization charts for the greedy and the uniform layout policies, respectively. In these figires, buffer underflow, which causes discontinuities during playback, are depicted by negative values of buffer occupancy. Consequently, the lowest negative values in the charts define the initial read ahead necessary to guarantee continuity of display, and the differences between the

www.manaraa.com

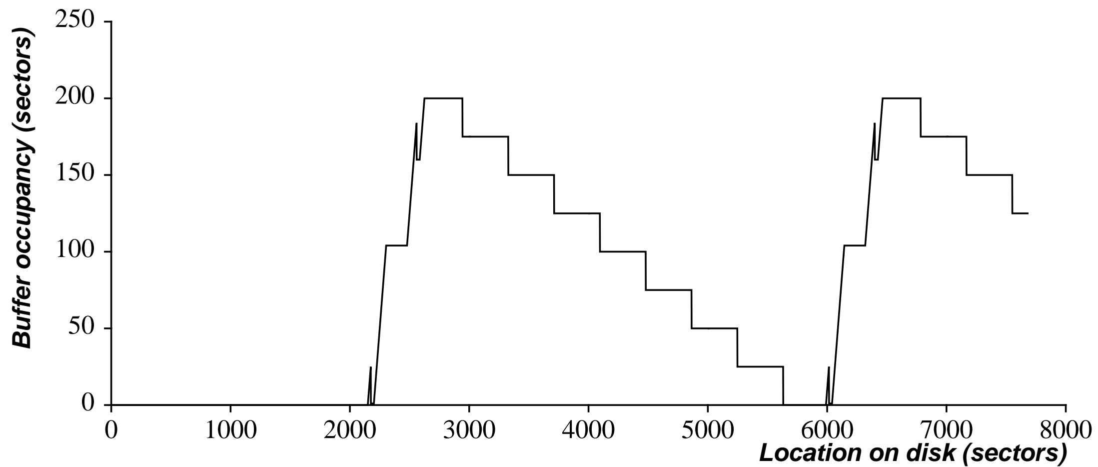

Figure 7: Buffer occupancy when the greedy layout policy is used in on-line merging

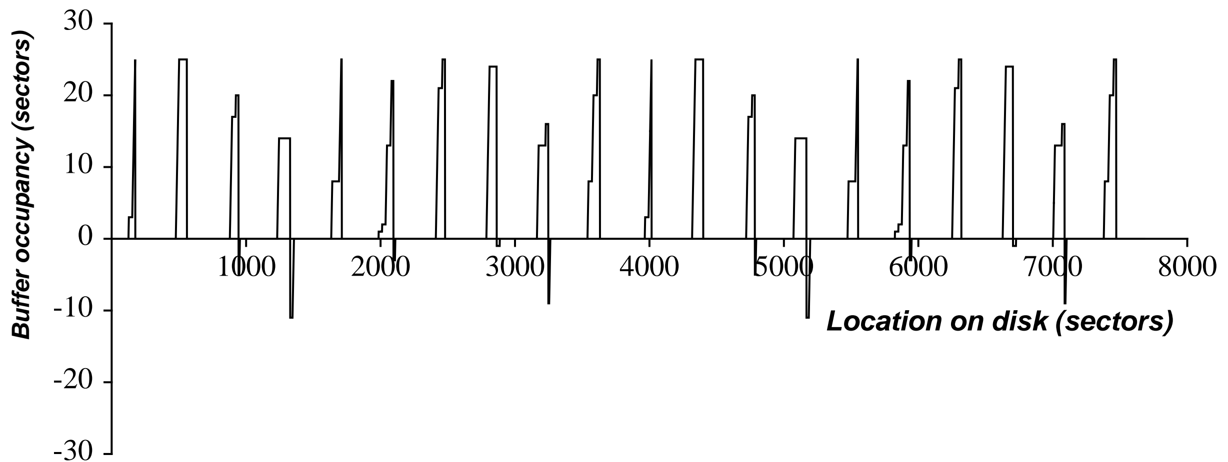

Figure 8: Buffer occupancy when the uniform layout policy is used in on-line merging

maximum positive value and the minimum negative value define the total buffer size needed. It can be seen that, whereas the greedy layout policy, which packs media blocks continuously starting from the very first gap, requires no read ahead but a buffer size of 200 sectors (Figure 7), the uniform layout policy, which uniformly distributes the media blocks over all the gaps, requires a read ahead of about 10 sectors and a buffer size of only 35 sectors (Figure 8). Clearly, the uniform layout policy yields a large reduction in the buffering requirements as compared to the greedy policy.

For the off-line algorithm, assuming the availability of two buffers for each strand (one to hold the block being transferred and the other for the block being displayed), the values of k_i obtained by the scaled placement policy for each of the six strands are shown in Table 3, yielding a total merge cycle length of 120 sectors, out of which,

المنارة للاستشارات
www.manaraa.com

Video strand recording rates (frames/sec)	Block size M (sectors)	Gap size G (sectors)	Pattern $M+G$ (sectors)	k_i's determined using scaled placement	Sequences of deadline (1's) and non-deadline (0's) merge cycles
32	25	95	120	1	$\{1, 1, 1, ...\}$
30	25	103	128	0.9375	$\{1, 0, 1, 1, ..., 1, 0, ...\}$
20	25	167	192	0.625	$\{1, 0, 0, 1, 1, 0, 1, 1, ...\}$
16	25	215	240	0.5	$\{1, 0, 0, 1, 0, 1, 0, 1, ...\}$
15	25	231	256	0.46875	$\{0, 0, 1, 0, 1, 0, 1, ...\}$
10	25	359	384	0.3125	$\{0, 0, 0, 1, 0, 0, 1, ...\}$

Table 3: Off-line merging using the scaled placement policy

23 sectors are unoccupied, resulting in a storage efficiency of 80 %, which is same as that of the on-line algorithm. Transforming the same k_i into integral values results in the sequences of deadline merge cycles shown in the last column of Table 3 ("1" in the sequence represents a deadline merge cycle, and "0" represents a non-deadline merge cycle). Since the total number of buffers required is just two for each strand, the off-line algorithm yields a reduction in the buffering requirements. Furthermore, display can be initiated as soon as a media block is retrieved from the disk, thereby eliminating the need for any read-ahead.

6 Concluding Remarks

We have addressed the problem of collocational storage of multiple media strands on disk. We have presented a model that relates disk and device characteristics to the playback rate of media strands, and derives storage patterns that can guarantee their continuous retrieval. To efficiently utilize the disk space, we have developed techniques for merging storage patterns of multiple media strands. We have proposed both an on-line algorithm suitable for merging a new media strand into a set of already stored strands, and an off-line merging algorithm that can be applied a priori to the storage of a set of media strands before any of them have been stored on disk. The on-line algorithm uses uniform layout techniques to minimize read-ahead and buffering requirements. The off-line algorithm can operate with much smaller buffer sizes, and it does so by using a staggered toggling technique in which sizes of successive media blocks are fine tuned individually so as to together not exceed the available buffer size (without, of course, violating the playback rate requirements of any of the strands). These algorithms are being implemented in the multimedia storage server being developed at the UCSD Multimedia Laboratory. Initial

المنارة للاستشارات
www.manaraa.com

performance estimations demonstrate significant gains in storage space utilization as a result of using both on-line and off-line merging algorithms. We are extending the techniques to handle variable block sizes and finite strands, as well.

References

[1] C. Abbott. Efficient Editing of Digital Sound on Disk. *Journal of Audio Engineering*, 32(6):394–402, June 1984.

[2] P. B. Berra, C. Y. R. Chen, A. Ghafoor, C. C. Lin, T. D. C. Little, and D. Shin. An Architecture for Distributed Multimedia Database Systems. *Computer Communications*, 8(3):413–427, April 1990.

[3] E. A. Fox. The Coming Revolution in Interactive Digital Video. *Communications of the ACM*, 7(32):794–801, July 1989.

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

[7] Y. Mori. Multimedia Real-Time File System. Technical report, Matshushita Electric Industrial Co., February 1990.

[8] 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.

[9] P. Venkat Rangan and D. C. Swinehart. Software Architecture for Integration of Video Services in the Etherphone Environment. *IEEE Journal on Selected Areas in Communication*, 9(9):1395–1404, December 1991.

[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, Harrick M. Vin, and Srinivas Ramanathan. Designing an On-Demand Multimedia Service. *IEEE Communications Magazine*, 30(7):56–65, July 1992.

[12] W. D. Sincoskie. System Architecture for a Large Scale Video on Demand Service. *Computer Networks and ISDN Systems, North-Holland*, 22:155–162, 1991.

[13] C. Yu, W. Sun, D.Bitton, Q. Yang, R. Bruno, and J. Yus. Efficient Placement of Audio Data on Optical Disks for Real-Time Applications. *Communications of the ACM*, 7(32):862–871, July 1989.

المنارة للاستشارات

www.manaraa.com