Multimedia Tools and Applications 5, 79–108 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Mitra: A Scalable Continuous Media Server*

SHAHRAM GHANDEHARIZADEH shahram@research.panasonic.com
ROGER ZIMMERMANN, WEIFENG SHI AND REZA REJAIE
DOUG IERARDI ierardi@cs.usc.edu
TA-WEI LI
Computer Science Department, University of Southern California, Los Angeles, California 90089

Abstract. Mitra is a scalable storage manager that supports the display of continuous media data types, e.g., audio and video clips. It is a software based system that employs off-the-shelf hardware components. Its present hardware platform is a cluster of multi-disk workstations, connected using an ATM switch. Mitra supports the display of a mix of media types. To reduce the cost of storage, it supports a hierarchical organization of storage devices and stages the frequently accessed objects on the magnetic disks. For the number of displays to scale as a function of additional disks, Mitra employs staggered striping. It implements three strategies to maximize the number of simultaneous displays supported by each disk. First, the EVEREST file system allows different files (corresponding to objects of different media types) to be retrieved at different block size granularities. Second, the FIXB algorithm recognizes the different zones of a disk and guarantees a continuous display while harnessing the average disk transfer rate. Third, Mitra implements the Grouped Sweeping Scheme (GSS) to minimize the impact of disk seeks on the available disk bandwidth.

In addition to reporting on implementation details of Mitra, we present performance results that demonstrate the scalability characteristics of the system. We compare the obtained results with theoretical expectations based on the bandwidth of participating disks. Mitra attains between 65% to 100% of the theoretical expectations.

Keywords: video, zoned disks, scalable servers, multi-disk hardware platforms, striping, file system management

1. Introduction

The past few years have witnessed many design studies describing different components of a server that supports continuous media data types, such as audio and video. The novelty of these studies is attributed to two requirements of continuous media that are different from traditional textual and record-based data. First, the retrieval and display of continuous media are subject to real-time constraints that impact both (a) the storage, scheduling and delivery of data, and (b) the manner in which multiple users may share resources. If the resources are not shared and scheduled properly then a display might starve for data, resulting in disruptions and delays that translate into jitter with video and random noises with audio. These disruptions and delays are termed *hiccups*. Second, objects of this media type are typically large in size. For example, a two hour MPEG-2 encoded video requiring 4 Megabits per second (Mbps) for its display is 3.6 Gigabyte in size. Three minutes of uncompressed CD quality audio with a 1.4 Mbps bandwidth requirement is 31.5 Megabyte

*This research was supported in part by a Hewlett-Packard unrestricted cash/equipment gift, and the National Science Foundation under grants IRI-9203389 and IRI-9258362 (NYI award).

(MByte) in size. The same audio clip in MPEG-encoded format might require 0.38 Mbps for its display and is 8.44 Mbyte.

Mitra is a realization of several promising design concepts described in the literature. Its primary contributions are two-fold: (1) to demonstrate the feasibility of these designs, and (2) to achieve the non-trivial task of gluing these together into a system that is both high performance and scalable. Mitra is a software based system that can be ported to alternative hardware platforms. It **guarantees** simultaneous display of a collection of different media types as long as the bandwidth required by the display of each media type is constant (isochronous). For example, Mitra can display both CD-quality audio clips with a 1.34 Mbps bandwidth requirement and MPEG-2 encoded streams with 4 Mbps bandwidth requirement (two different media types) at the same time as long as the bandwidth required by each display is constant. Moreover, Mitra can display those media types whose bandwidth requirements might exceed that of a single disk drive (e.g., uncompressed NTSC CCIR 601 video clips requiring 270 Mbps for their display) in support of high-end applications that cannot tolerate the use of compression techniques.

Due to their large size, continuous media objects are almost always disk resident. Hence, the limiting resource in Mitra is the available disk bandwidth, i.e., traditional I/O bottleneck phenomena. Mitra is scalable because it can service a higher number of simultaneous displays as a function of additional disk bandwidth. The key technical idea that supports this functionality is to distribute the workload imposed by each display evenly across the available disks using staggered striping [3] to avoid the formation of hot spots and bottleneck disks.

Mitra is high performance because it implements techniques that maximize the number of displays supported by each disk. This is accomplished in two ways. First, Mitra minimizes both the number of seeks incurred when reading a block (using EVEREST [18]) and the amount of time attributed to each seek (using GSS [33]). Second, it maximizes the transfer rate of multi-zone disks by utilizing the bandwidth of different zones in an intelligent manner (FIXB [19]). Mitra's file system is EVEREST. As compared with other file systems, EVEREST provides two functionalities. First, it enables Mitra to retrieve different files at different block size granularities. This minimizes the percentage of disk bandwidth that is wasted when Mitra displays objects that have different bandwidth requirements. Second, it avoids the fragmentation of disk space when supporting a hierarchy of storage devices [4] where different objects are swapped in and out of the available disk space over time. GSS minimizes the amount of time attributed to each seek by optimizing the disk scheduling algorithm. Finally, FIXB in combination with EVEREST enables Mitra to guarantee a continuous display while harnessing the average transfer rate of multi-zone disks [31, 16]. FIXB enables Mitra to strike a compromise between the percentage of wasted disk space and how much of its transfer rate is harnessed. With each of these techniques, there are tradeoffs associated with the choices of values for system parameters. Although these tradeoffs have been investigated using analytical and simulation studies, Mitra's key contribution is to demonstrate that these analyses hold true in practice. It shows that one does not have to rewrite software to support diverse applications with different performance objectives (startup latency versus throughput versus wasted disk space). Instead, there is a single system, where different choices of parameters support different application requirements.

Table 1. Parameters and their definition.

Parameter	Definition
η	Number of media types
$\mathcal{R}_C(M_i)$	Bandwidth required to display objects of media type i
$\mathcal{R}_D$	Bandwidth of a disk
$\mathcal{B}(M_i)$	Block size for media type i
D	Total number of disks
d	Number of disks that constitute a cluster
$\mathcal{C}$	Number of clusters recognized by the system
g	Number of groups with GSS
k	Stride with staggered striping
$\mathcal{N}$	Number of simultaneous displays supported by the system
S	Maximum height of sections with EVEREST
ω	Number of contiguous buddies of section height i that form a section of height $i + 1$

Several related studies have described the implementation of continuous media servers[1]. These can be categorized into single-disk and multi-disk systems. The single-disk systems include [1, 8, 26, 30]. These pioneering studies were instrumental in identifying the requirements of continuous media. They developed scheduling policies for retrieving blocks from disk into memory to support a continuous display. (Mitra employs these policies as detailed in Section 3.) Compared with Mitra, most of them strive to be general purpose and support traditional file system accesses in addition to a best-effort delivery of continuous media. Thus, none strive to maximize the number of displays supported by a disk using alternative disk scheduling policies, techniques that harness the average transfer rate of disk zones, or strategies that constrained the physical file layout. The multi-disk systems include: Streaming RAID [32], Fellini [28], and Minnesota's VOD server [23]. None claims to support either the display of a mix of media types or a hierarchical storage structure, nor do they describe the implementation of a file system that ensures contiguous layout of a block on the disk storage medium. (The authors of Fellini identify the design of a file system such as the one developed for Mitra as an important research direction in [29].) Moreover, all three systems employ disk arrays where the number of disks that are treated as a single logical disk is pre-determined by the hardware. Mitra differs in that the number of disks that are treated as one logical disk is NOT hardware dependent. Instead, it is determined by the bandwidth requirement of a media type. Indeed, if one analyzes two different displays with each accessing a different media type, one display might treat two disks as one logical disk while the other might treat five disks as one logical disk. This has a significant impact on the number of simultaneous displays supported by the system as detailed in Section 4.

Streaming RAID implements GSS to maximize the bandwidth of a disk array and employs memory-sharing to minimize the amount of memory required at the server. It develops analytical models similar to [12, 14] to estimate the performance of the system with alternative configuration parameters. Fellini analyzes constraint placement of data to enhance

Table 2. Defining terms.

Mbps	Megabits per second
Block	Amount of data retrieved per time period on behalf of a PM displaying an object of media type i. Its size varies depending on the media type and is denoted as $\mathcal{B}(M_i)$
Fragment	Fraction of a block assigned to one disk of a cluster that contains the block. All fragments of a block are equi-sized
Time period	The amount of time required to display a block at a station. This time is fixed for all media types, independent of their bandwidth requirement
Page	Basic unit of allocation with EVEREST, also termed sections of height 0
Startup latency	Amount of time elapsed from when a PM issues a request for an object to the onset of the display

the performance of the system with multi-zone disks. The design appears to be similar to FIXB. Fellini describes several designs to support VCR features such as Fast Forward and Rewind. (We hint at Mitra's designs to support this functionality in Section 5 and do not detail them due to lack of space.) Neither Fellini nor Streaming RAID present performance numbers from their system. Minnesota's VOD server differs from both Mitra and the other two multi-disk systems in that it does not have a centralized scheduler. Hence, it cannot guarantee a continuous display. However, [23] presents performance numbers to demonstrate that a mass storage system can display continuous media.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the software components of Mitra and its current hardware platform. Section 3 describes the alternative components of the system (EVEREST, GSS, FIXB, and staggered striping) and how they interact with each other to guarantee a continuous display. Section 4 presents experimental performance results from Mitra. As a yard stick, we compare these numbers with theoretical expectations based on the available disk bandwidth [14, 15]. The obtained results: (1) demonstrate the scalability of the system, (2) show that Mitra attains between 65% to 100% of the theoretical expectations. Our future research directions are presented in Section 5.

2. An overview of Mitra

Mitra employs a hierarchical organization of storage devices to minimize the cost of providing on-line access to a large volume of data. It is currently operational on a cluster of HP 9000/735 workstations. It employs a HP Magneto Optical Juke-box as its tertiary storage device. Each workstation consists of a 125 MHz PA-RISC CPU, 80 MByte of memory, and four Seagate ST31200W magnetic disks. Mitra employs the HP-UX operating system (version 9.07) and is portable to other hardware platforms. While 15 disks can be attached to the fast and wide SCSI-2 bus of each workstation, we attached four disks to this chain because additional disks would exhaust the bandwidth of this bus. It is undesirable to exhaust the bandwidth of the SCSI-2 bus for several reasons. First, it would cause the underlying hardware platform to not scale as a function of additional disks. Mitra is a software system and if its underlying hardware platform does not scale then the entire system would not scale. Second, it renders the service time of each disk unpredictable, resulting in hiccups.

Mitra consists of three software components:

1. Scheduler: This component schedules the retrieval of the blocks of a referenced object in support of a hiccup-free display at a PM. In addition, it manages the disk bandwidth and performs admission control. Currently, Scheduler includes an implementation of EVEREST, staggered striping, and techniques to manage the tertiary storage device. It also has a simple relational storage manager to insert, and retrieve information from a *catalog*. For each media type, the catalog contains the bandwidth requirement of that media type and its block size. For each presentation, the catalog contains its name, whether it is disk resident (if so, the name of EVEREST files that represent this clip), the cluster and zone that contains its first block, and its media type.
2. Mass storage Device Manager (DM): Performs either disk or tertiary read/write operations.
3. Presentation Manager (PM): Displays either a video or an audio clip. It might interface with hardware components to minimize the CPU requirement of a display. For example, to display an MPEG-2 clip, the PM might employ either a program or a hardware-card to decode and display the clip. The PM implements the PM-driven scheduling policy of Section 3.1.3 to control the flow of data from the Scheduler.

Mitra uses UDP for communication between the process instantiation of these components. UDP is an unreliable transmission protocol. Mitra implements a light-weight kernel, named HP-NOSE. HP-NOSE supports a window-based protocol to facilitate reliable transmission of messages among processes. In addition, it implements the threads with shared memory, ports that multiplex messages using a single HP-UX socket, and semaphores for synchronizing multiple threads that share memory. An instantiation of this kernel is active per Mitra process.

For a given configuration, the following processes are active: one Scheduler process, a DM process per mass storage read/write device, and one PM process per active client. For example, in our twelve disk configuration with a magneto optical juke box, there are sixteen active processes: fifteen DM processes, and one Scheduler process (see figure 1). There are two active DM processes for the magneto juke-box because it consists of two read/write devices (and 32 optical platters that might be swapped in and out of these two devices).

The combination of the Scheduler with DM processes implements asynchronous read/ write operations on a mass storage device (which is otherwise unavailable with HP-UX 9.07). This is achieved as follows. When the Scheduler intends to read a block from a device (say a disk), it sends a message to the DM that manages this disk to read the block. Moreover, it requests the DM to transmit its block to a destination port address (e.g., the destination might correspond to the PM process that displays this block) and issue a done message to the Scheduler. There are several reasons for not routing data blocks to active PMs using the Scheduler. First, it would waste the network bandwidth with multiple transmissions of a block. Second, it would limit the scalability of the system because the processing capability of the workstation that supports the Scheduler process would determine the overall throughput of the system. CPU processing is required because a transmitted data block

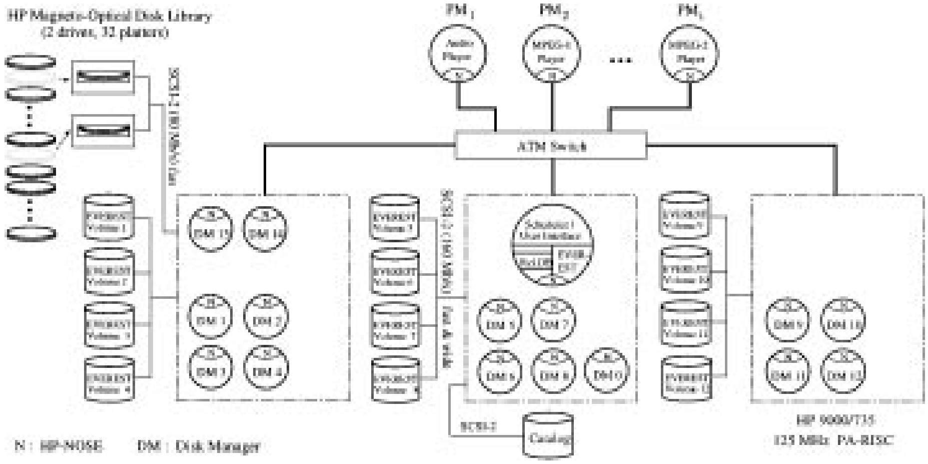


Figure 1. Hardware and software organization of Mitra. Note: While 15 disks can be attached to the fast and wide SCSI-2 bus of each workstation, we attached four disks because additional disks would exhaust the bandwidth of this bus.

is copied many times by different layers of software that implement the Scheduler process: HP-UX, HP-NOSE, and the Scheduler.

While the interaction between the different processes and threads is interesting, we do not report on them due to lack of space.

3. Continuous display with Mitra

We start by describing the implementation techniques of Mitra for a configuration that treats the d available disks as a single disk drive. This discussion introduces EVEREST [18], Mitra's file system, and motivates a PM-driven scheduling paradigm that provides feedback from a PM to the Scheduler to control the rate of data production. Subsequently, we discuss an implementation of the staggered striping [3] technique.

3.1. One disk configuration

To simplify the discussion and without loss of generality, conceptualize the d disks as a single disk with the aggregate transfer rate of d disks. When we state that a block is assigned to the disk, we imply that the block is declustered [3, 10] across the d disks. Each piece of this block is termed a *fragment*. Moreover, when we state a DM reads a block from the disk, we imply that d DM processes are activated simultaneously to produce the fragments that constitute the block.

To display an object X of media type M_i (say CD-quality audio) with bandwidth requirement $\mathcal{R}_C(M_i)$ (1.34 Mbps), Mitra conceptualizes X as consisting of r blocks: $X_0, X_1, \ldots, X_{r-1}$. Assuming a block size of $\mathcal{B}(M_i)$, the display time of a block, termed a *time period* [15], equals $\frac{\mathcal{B}(M_i)}{\mathcal{R}_C(M_i)}$. Assuming that the system is idle, when a PM references object X, the Scheduler performs two tasks. First, it issues a read request for X_0 to the DM. It also provides the network address of the PM, requesting the DM to forward X_0 directly to the PM. Second, after a pre-specified delay, it sends a control message to the PM to initiate the display of X_0. This delay is due to the implementation of both GSS [33] (detailed below) and FIXB [19] (described in Section 3.1.2). Once the PM receives a block, it waits for a control message from the Scheduler before initiating the display. The Scheduler requests the DM to transmit the next block of X (i.e., X_1) in the next time period to the PM. This enables the PM to provide for a smooth transition between the two blocks to provide for a hiccup-free display. With the current design, a PM requires enough memory to cache at least two blocks of data.

Given a database that consists of η different media types (say $\eta = 2$, MPEG-2 and CD-quality audio), the block size of each media type is determined such that the display time of a block (i.e., the duration of a time period at the Scheduler) is fixed for all media types. This is done as follows. First, one media type M_i (say CD-quality audio) with bandwidth requirement $\mathcal{R}_C(M_i)$ (1.34 Mbps) defines the base block size $\mathcal{B}(M_i)$ (say 512 KByte). The block size of other media types is a function of their bandwidth, $\mathcal{R}_C(M_i)$, and $\mathcal{B}(M_i)$. For each media type M_j, its block size is:

$$\mathcal{B}(M_j) = \frac{\mathcal{R}_C(M_j)}{\mathcal{R}_C(M_i)} \times \mathcal{B}(M_i)$$

In our example, the block size for MPEG-2 (4 Mbps) objects would be 1521.74 KByte.

In an implementation of a file system, the physical characteristics of a magnetic disk determines the granularity for the size of a block. With almost all disk manufacturers, the granularity is limited to $\frac{1}{2}$ KByte[2]. Mitra rounds up the block size of each object of a media type to the nearest $\frac{1}{2}$ KByte. Thus, in our example, the block size for MPEG-2 object would be 1522 KByte. However, Mitra does not adjust the duration of a time period to reflect this rounding up. Thus, for each time period, the system produces more data on behalf of a display as compared to the amount that the display consumes. The amount of accumulated data is dependent on both the number of blocks that constitute a clip and what fraction of each block is not displayed per time period. For example, with a two hour MPEG-2 video object, a display would have accumulated 622.7 KByte of data at the end of the display. Section 3.1.3 describes a scheduling paradigm that prevents the Scheduler from producing data should the amount of cached data become significant.

Mitra supports the display of $\mathcal{N}$ objects by multiplexing the disk bandwidth among $\mathcal{N}$ block retrievals. Its admission control policy ensures that the service time of these $\mathcal{N}$ block retrievals does not exceed the duration of a time period. The service time of the disk to retrieve a block of media type i is a function of $\mathcal{B}(M_i)$, the disk transfer rate, rotational latency, and seek time. Mitra opens each disk in RAW mode [22]. We used the SCSI commands to interrogate the physical characteristics of each disk to determine its track sizes, seek characteristics, number of zones, and transfer rate of each zone. (To gather this information, one requires neither specialized hardware nor the use of the assembly programming language, see [16] for a detailed description of these techniques.) The Scheduler reads this information from a configuration file during its startup.

The Scheduler maintains the duration of a time period using a global variable and supports a linked list of requests that are currently active. In addition to other information, an element of this list records the service time of the disk to retrieve a block of the file referenced by this display. Mitra minimizes the impact of seeks incurred when retrieving blocks of different objects by implementing the GSS algorithm. With GSS, a time period might be partitioned into g groups. In its simplest form, GSS is configured with one group ($g = 1$). With $g = 1$, a PM begins to consume the block that was retrieved on its behalf during time period ℓ at the beginning of time period $\ell + 1$. This enables the disk scheduling algorithm to minimize the impact of seeks by retrieving the blocks referenced during a time period using a scan policy. Mitra implements this by synchronizing the display of the first block of an object (X_0) at the PM with the end of the time period that retrieved X_0. Once the display of X_0 is synchronized, the display of the other blocks are automatically synchronized due to a fixed duration for each time period. The synchronization of X_0 is achieved as follows. A PM does not initiate the display of X_0 until it receives a control message from the Scheduler. The Scheduler generates this message at the beginning of the time period that retrieved X_1.

With $g > 1$, Mitra partitions a time period into g equi-sized intervals. The Scheduler assigns a display to a single group and the display remains with this group until its display is complete. The retrieval of blocks assigned to a single group employs the elevator scheduling algorithm. This is implemented as follows. Assuming that group G_i retrieves a block of X per time period, the display of X_0 is started when the disk subsystem begins to service group G_{i+1}.

3.1.1. File system design. The current implementation of Mitra assumes that a PM does not perform complex operations such as Fast-Forward, Fast-Rewind or Pause operations. Upon the arrival of a request for object X belonging to media type M_X, the admission control policy of the Scheduler is as follows. First, the Scheduler checks to see if another scheduled display is beginning the display of X, i.e., references X_0. If so, these two new requests are combined with each other into one. This enables Mitra to multiplex a single stream among multiple PMs. If no other stream is referencing X_0, starting with the current active group, the Scheduler locates the group with sufficient idle time to accommodate the retrieval of a block of size $\mathcal{B}(M_i)$. The implementation details of this policy are contained in Appendix A. If no group can accommodate the retrieval of this request, the Scheduler queues this request and examines the possibility of admitting it during the next time period.

With η media types, Mitra's file system might be forced to manage η different block sizes. Moreover, the blocks of different objects might be staged from the tertiary storage device onto magnetic disk storage on demand. A block should be stored contiguously on disk. Otherwise, the disk would incur seeks when reading a block, reducing disk bandwidth. Moreover, it might result in hiccups because the retrieval time of a block might become unpredictable. To ensure a contiguous layout of a block, we considered four alternative approaches: disk partitioning, extent-based [2, 5, 21], multiple block sizes, and an approximate contiguous layout of a file. We chose the final approach, resulting in the design and implementation of the EVEREST file system. Below, we describe each of the other three approaches and our reasons for abandoning them.

With disk partitioning, assuming η media types with η different block sizes, the available disk space is partitioned into η regions, one region per media type. A region i corresponds to media type i. The space of this region is partitioned into fix sized blocks, corresponding to $\mathcal{B}(M_i)$. The objects of media type i compete for the available blocks of this region. The amount of space allocated to a region i might be estimated as a function of both the size and frequency of access of objects of media type i [13]. However, partitioning of disk space is inappropriate for a dynamic environment where the frequency of access to the different media types might change as a function of time. This is because when a region becomes cold, its space should be made available to a region that has become hot. Otherwise, the hot region might start to exhibit a thrashing [6] behavior that would increase the number of retrievals from the tertiary storage device. This motivates a re-organization process to re-arrange disk space. This process would be time consuming due to the overhead associated with performing I/O operations.

With an extent-based design, a fixed contiguous chunk of disk space, termed an extent, is partitioned into fix-sized blocks. Two or more extents might have different page sizes. Both the size of an extent and the number of extents with a pre-specified block size (i.e., for a media type) is fixed at system configuration time. A single file may span one or more extents. However, an extent may contain no more than a single file. With this design, an object of a media type i is assigned one or more extents with block size $\mathcal{B}(M_i)$. In addition to suffering from the limitations associated with disk partitioning, this approach suffers from internal fragmentation with the last extent of an object being only partially occupied. This would waste disk space, increasing the number of references to the tertiary storage device.

With the Multiple Bock Size approach (MBS), the system is configured based on the media type with the lowest bandwidth requirement, say M_1. MBS requires the block size of each of media type j to be a multiple of $\mathcal{B}(M_1)$, i.e., $\mathcal{B}(M_j) = \lceil \frac{\mathcal{B}(M_j)}{\mathcal{B}(M_1)} \rceil \mathcal{B}(M_1)$. This might simplify the management of disk space to: 1) avoid its fragmentation, and 2) ensure the contiguous layout of each block of an object. However, MBS might waste disk bandwidth by forcing the disk to: (1) retrieve more data on behalf of a PM per time period due to rounding up of block size, and (2) remain idle during other time periods to avoid an overflow of memory at the PM. These are best illustrated using an example. Assume two media types MPEG-1 and MPEG-2 objects with bandwidth requirements of 1.5 Mbps and 4 Mbps, respectively. With this approach, the block size of the system is chosen based on MPEG-1 objects. Assume, it is chosen to be 512 KByte, $\mathcal{B}$(MPEG-1) = 512 KByte. This implies that $\mathcal{B}$(MPEG-2) = 1365.33 KByte. MBS would increase $\mathcal{B}$(MPEG-2) to equal 1536 KByte. To avoid excessive amount of accumulated data at a PM displaying an MPEG-2 clip, the Scheduler might skip the retrieval of data one time period every nine time periods using the PM-driven scheduling paradigm of Section 3.1.3. The Scheduler may not employ this idle slot to service another request because it is required during the next time period to retrieve the next block of current MPEG-2 display. If all active requests are MPEG-2 video clips and a time period supports nine displays with $\mathcal{B}$(MPEG-2) = 1536 KByte then, with $\mathcal{B}$(MPEG-2) = 1365.33 KByte, the system would support ten simultaneous displays (10% improvement in performance). In summary, the block size for a media type should approximate its theoretical value in order to maximize the number of simultaneous displays.

The final approach, and the one used by Mitra, employs the buddy algorithm to approximate a contiguous layout of a file on the disk without wasting disk space. The number of contiguous chunks that constitute a file is a fixed function of the file size and the configuration of the buddy algorithm. Based on this information, Mitra can either (1) prevent a block from overlapping two non-contiguous chunks or (2) allow a block to overlap two chunks and require the PM to cache enough data to hide the seeks associated with the retrieval of these blocks. Currently, Mitra implements the first approach. To illustrate the second approach, if a file consists of five contiguous chunks then at most four blocks of this file might span two different chunks. This implies that the retrieval of four blocks will incur seeks with at most one seek per block retrieval. To avoid hiccups, the Scheduler should delay the display of the data at the PM until it has cached enough data to hide the latency associated with four seeks. The amount of cached data is not significant. For example, assuming a maximum seek time of 20 milliseconds, with MPEG-2 objects (4 Mbps), the PM should cache 10 KByte to hide each seek. However, this approach complicates the admission control policy because the retrieval of a block might incur either one or zero seeks.

EVEREST. With EVEREST, the basic unit of allocation is a page,[3] also termed *sections* of height 0. EVEREST organizes these sections as a tree to form larger, contiguous sections. As illustrated in figure 2, only sections of size(page) × ω^i (for $i \geq 0$) are valid, where the base ω is a system configuration parameter. If a section consists of ω^i pages then i is said to be the height of the section. The system can combine ω height i sections that are buddies (physically adjacent) to construct a section of height $i + 1$.

To illustrate, the disk in figure 2 consists of 16 pages. The system is configured with $\omega = 2$. Thus, the size of a section may vary from 1, 2, 4, 8, up to 16 pages. In essence, a binary tree is imposed upon the sequence of pages. The maximum height, computed by[4] $S = \lceil \log_\omega(\lfloor \frac{\text{Capacity}}{\text{size(page)}} \rfloor) \rceil$, is 4. With this organization imposed upon the device, sections of height $i \geq 0$ cannot start at just any page number, but only at offsets that are multiples of ω^i. This restriction ensures that any section, with the exception of the one at height S, has a total of $\omega - 1$ adjacent *buddy* sections of the same size at all times. With the base 2 organization of figure 2, each section has one buddy.

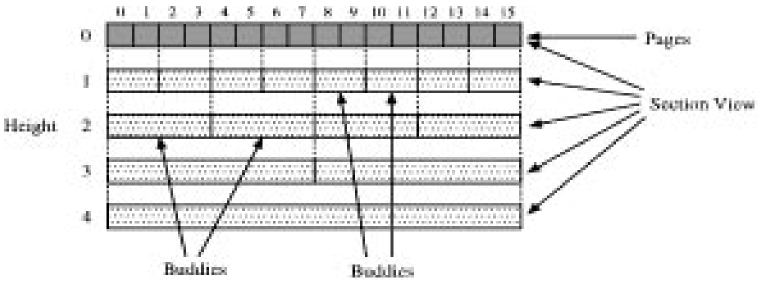


Figure 2. Physical division of disk space into pages and the corresponding logical view of the sections with an example base of $\omega = 2$.

With EVEREST, a portion of the available disk space is allocated to objects. The remainder, should any exist, is free. The sections that constitute the available space are handled by a *free list*. This free list is actually maintained as a sequence of lists, one for each section height. The information about an unused section of height i is enqueued in the list that handles sections of that height. In order to simplify object allocation, the following *bounded list length property* is always maintained: For each height $i = 0, \ldots, S$, at most $\omega - 1$ free sections of i are allowed. Informally, this property implies that whenever there exists sufficient free space at the free list of height i, EVEREST *must* compact these free sections into sections of a larger height[5].

The process of staging an object from tertiary onto available disk space is as follows. The first step is to check, whether the total number of pages in all the sections on the free list is either greater than or equal to the number of pages (denoted no-of-pages(X)) that the new object X requires. If this is not the case then one or more victim objects are elected and deleted. (The procedure for selecting a victim is based on heat [17]. The deletion of a victim object is described further below.) Assuming enough free space is available at this point, X is divided into its corresponding sections as follows. First, the number $m =$ no-of-pages(X) is converted to base ω. For example, if $\omega = 2$, and no-of-pages$(X) = 13_{10}$ then its binary representation is 1101_2. The full representation of such a converted number is $m = d_{j-1} \times \omega^{j-1} + \cdots + d_2 \times \omega^2 + d_1 \times \omega^1 + d_0 \times \omega^0$. In our example, the number 1101_2 can be written as $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$. In general, for every digit d_i that is non-zero, d_i sections are allocated from height i of the free list on behalf of X. In our example, X requires 1 section from height 0, no sections from height 1, 1 section from height 2, and 1 section from height 3.

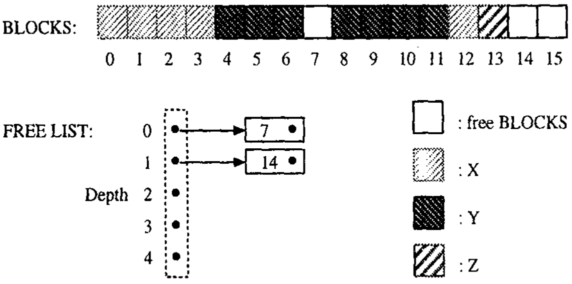
For each object, the number ν of contiguous pieces is equal to the number of one's in the binary representation of m, or with a general base ω, $\nu = \sum_{i=0}^{j} d_i$ (where j is the total number of digits). Note that ν is always bounded by $\omega \lceil \log_\omega m \rceil$. For any object, ν defines the maximum number of sections occupied by the object. (The minimum is 1 if all ν sections are physically adjacent.) A complication arises when no section at the right height exists. For example, suppose that a section of size ω^i is required, but the smallest section larger than ω^i on the free list is of size ω^j $(j > i)$. In this case, the section of size ω^j can be split into ω sections of size ω^{j-1}. If $j - 1 = i$, then $\omega - 1$ of these are enqueued on the list of height i and the remainder is allocated. However, if $j - 1 > i$ then $\omega - 1$ of these sections are again enqueued at level $j - 1$, and the splitting procedure is repeated on the remaining section. It is easy to see that, whenever the total amount of free space on these lists is sufficient to accommodate the object, then for each section that the object occupies, there is always a section of the appropriate size, or larger, on the list. This splitting procedure will guarantee that the appropriate number of sections, each of the right size, will be allocated, and that the bounded list length property is never violated.

When there is insufficient free disk space to materialize an object, then one or more victim objects (with copies on tertiary) are removed from the disk. Reclaiming the space of a victim requires two steps for each of its sections. First, the section must be appended to the free list at the appropriate height. The second step ensures that the bounded list length property is not violated. Therefore, whenever a section is enqueued in the free list at height i and the number of sections at that height is equal to or greater than ω, then ω sections must be combined into one section at height $i + 1$. If the list at $i + 1$ now violates bounded list length property, then once again space must be compacted and moved to section $i + 2$. This procedure might be repeated several times. It terminates when the length of the list for a higher height is less than ω.

Compaction of ω free sections into a larger section is simple when they are buddies; in this case, the combined space is already contiguous. Otherwise, the system might be forced to exchange one occupied section of an object with one on the free list in order to ensure contiguity of an appropriate sequence of ω sections at the same height. The following algorithm achieves space-contiguity among ω free sections at height i.

1. Check if there are at least ω sections for height i on the free list. If not, stop.
2. Select the first section (denoted s_j) and record its page-number (i.e., the offset on the disk drive). The goal is to free $\omega - 1$ sections that are buddies of s_j.
3. Calculate the page-numbers of s_j's buddies. EVEREST's division of disk space guarantees the existence of $\omega - 1$ buddy sections physically adjacent to s_j.
4. For every buddy s_k, $k \leq 0 \leq \omega - 1$, $k \neq j$, if it exists on the free list then mark it.
5. Any of the s_k unmarked buddies currently store parts of other object(s). The space must be re-arranged by swapping these s_k sections with those on the free list. Note that for every buddy section that should be freed there exists a section on the free list. After swapping space between every unmarked buddy section and a free list section, enough contiguous space has been acquired to create a section at height $i + 1$ of the free list.
6. Go back to Step 1.

To illustrate, consider the organization of space in figure 3(a). The initial set of disk resident objects is $\{X, Y, Z\}$ and the system is configured with $\omega = 2$. In figure 3(a), two sections are on the free list at height 0 and 1 (addresses 7 and 14 respectively), and Z is



(a)

Figure 3. Deallocation of an object. The example sequence shows the removal of object Z from the initial disk resident object set $\{X, Y, Z\}$. Base two, $\omega = 2$. (a) Two sections are on the free list already (7 and 14) and object Z is deallocated. (b) Sections 7 and 13 should be combined, however they are not contiguous. (c) The buddy of section 7 is 6. Data must move from 6 to 13. (d) Sections 6 and 7 are contiguous and can be combined. (e) The buddy of section 6 is 4. Data must move from (4, 5) to (14, 15). (f) Sections 4 and 6 are now adjacent and can be combined. (g) The final view of the disk and the free list after removal of Z.

(*Continued on next page.*)

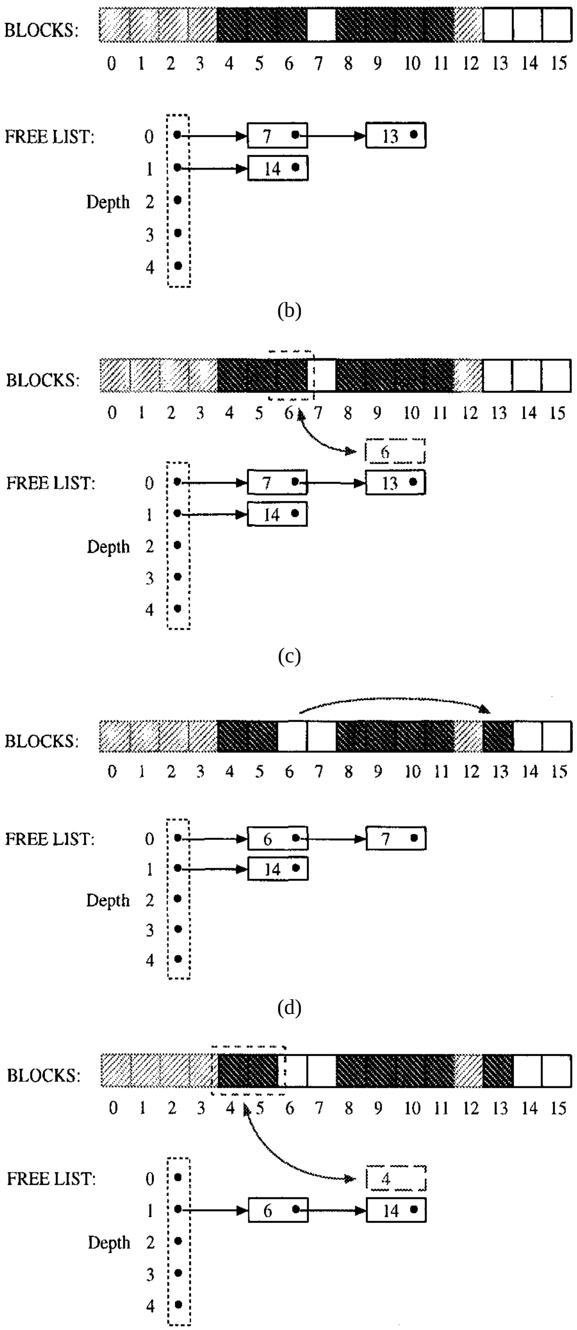


Figure 3. (*Continues.*)

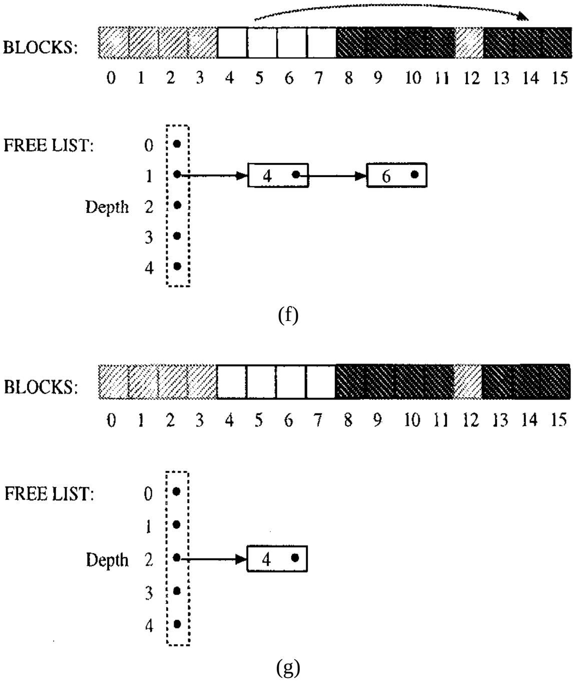


Figure 3. (*Continued.*)

the victim object that is deleted. Once page 13 is placed on the free list in figure 3(b), the number of sections at height 0 is increased to ω and it must be compacted according to Step 1. As sections 7 and 13 are not contiguous, section 13 is elected to be swapped with section 7's buddy, i.e., section 6 (figure 3(c)). In figure 3(d), the data of section 6 is moved to section 13 and section 6 is now on the free list. The compaction of sections 6 and 7 results in a new section with address 6 at height 1 of the free list. Once again, a list of length two at height 1 violates the bounded list length property and pages (4, 5) are identified as the buddy of section 6 in figure 3(e). After moving the data in figure 3(f) from pages (4, 5) to (14, 15), another compaction is performed with the final state of the disk space emerging as in figure 3(g).

Once all sections of a deallocated object are on the free list, the iterative algorithm above is run on each list, from the lowest to the highest height. The previous algorithm is somewhat simplified because it does not support the following scenario: a section at height i is not on the free list, however, it has been broken down to a lower height (say $i - 1$) and not all subsections have been used. One of them is still on the free list at height $i - 1$. In these cases, the free list for height $i - 1$ should be updated with care because those free sections have moved to new locations. In addition, note that the algorithm described above actually performs more work than is strictly necessary. A single section of a small height, for example, may end up being read and written several times as its section is combined into larger and larger sections. This is eliminated in the following manner. The algorithm is first performed “virtually”—that is, in main memory, as a compaction algorithm on the free lists. Once completed, the entire sequence of operations that have been performed determines the ultimate destination of each of the modified sections. The Scheduler constructs a list of these sections. This list is inserted into a queue of house keeping I/Os. Associated with each element of the queue is an estimated amount of time required to perform the task. Whenever the Scheduler locates one or more idle slots in the time period, it analyzes the queue of work for the element that can be processed using the available time. (Idle slots might be available with a workload that has completely utilized the number of idle slots due to the PM-driven scheduling paradigm of Section 3.1.3.)

The value of ω impacts the frequency of preventive operations. If ω is set to its minimum value (i.e., $\omega = 2$), then preventive operations would be invoked frequently because every time a new section is enqueued there is a 50% chance for a height of the free list to consist of two sections (violates the bounded list length property). Increasing the value of ω will therefore “relax” the system because it reduces the probability that an insertion to the free list would violate the bounded list length property. However, this would increase the expected number of bytes migrated per preventive operation. For example, at the extreme value of $\omega = n$ (where n is the total number of pages), the organization of blocks will consist of two levels, and for all practical purpose, EVEREST reduces to a standard file system that manages fix-sized pages.

The design of EVEREST suffers from the following limitation: the overhead of its preventive operations may become significant if many objects are swapped in and out of the disk drive. This occurs when the working set of an application cannot become resident on the disk drive.

In our implementation of EVEREST, it was not possible to fix the number of disk pages as an exact power of ω. The most important implication of an arbitrary number of pages is that some sections may not have the correct number of buddies ($\omega - 1$ of them). However, we can always move those sections to one end of the disk—for example, to the side with the highest page-offsets. Then instead of choosing the first section in Step 2 in the object deallocation algorithm, Mitra chooses the one with the lowest page-number. This ensures that the sections towards the critical end of the disk—that might not have the correct number of buddies—are never used in both Steps 4 and 5 of the algorithm.

Our implementation enables a process to retrieve a file using block sizes that are at the granularity of $\frac{1}{2}$ KByte. For example, EVEREST might be configured with a 64 KByte page size. One process might read a file at the granularity of 1365.50 KByte blocks, while another might read a second file at the granularity of 512 KByte.

The design of EVEREST is related to the buddy system proposed in [24, 25] for an efficient main memory storage allocator (DRAM). The difference is that EVEREST satisfies a request for b pages by allocating a number of sections such that their total number of pages equals b. The storage allocator algorithm, on the other hand, will allocate *one* section that is rounded up to $2^{\lceil lg\, b \rceil}$ pages, resulting in fragmentation and motivating the need for either a re-organization process or a garbage collector [21]. The primary advantage of the elaborate object deallocation technique of EVEREST is that it avoids both internal and external fragmentation of space as described for traditional buddy systems (see [21]).

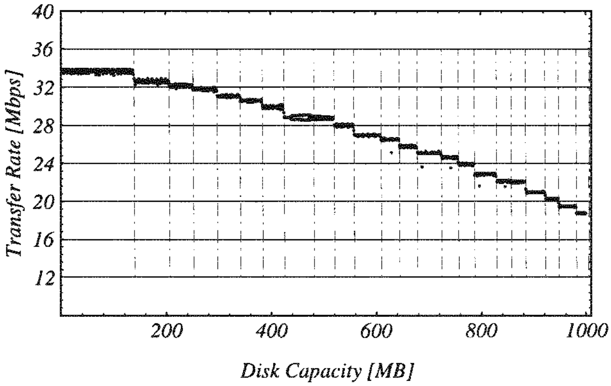


Figure 4. Zone characteristics of the seagate ST31200W magnetic disk.

3.1.2. Multi-zone disks. A trend in the area of magnetic disk technology is the concept of *zoning*. It increases the storage capacity of each disk. However, it results in a disk with variable transfer rates with different regions of the disk providing different transfer rates. Figure 4 shows the transfer rate of the 23 different zones that constitute each of the Seagate disks. (Techniques employed to gather these numbers are reported in [16].)

A file system that does not recognize the different zones might be forced to assume the bandwidth of the slowest zone as the overall transfer rate of the disk in order to guarantee a continuous display. In [19], we described two alternative techniques to support continuous display of audio and video objects using multi-zone disks, namely, FIXed Block size (FIXB) and VARiable Block size (VARB). These two techniques harness the average transfer rate of zones. Mitra currently implements FIXB[6]. It organizes an EVEREST file system on each region of the disk drive. Next, it assigns the blocks of each object to the zones in a round-robin manner. The blocks of each object that are assigned to a zone are stored as a single EVEREST file. In the catalog, Mitra maintains the identity of each EVEREST file that constitute a clip, its block size, and the zone that contains the first block of this clip.

The Scheduler scans the disk in one direction, say starting with the outermost zone moving inward. It recognizes m different zones, however, only one zone is active per time period. A global variable Z_{Active} denotes the identity of the active zone. The bandwidth of each zone is multiplexed among all active displays. Once the disk reads data from the innermost zone, it is repositioned to the outermost zone to start another sweep. The time to perform on weep is denoted as T_{scan}. The block size is chosen such that the amount of data produced by Mitra for a PM during one T_{scan} equals the amount of data consumed at the PM. This requires the faster zones to compensate for the slower zones. As demonstrated in figure 5, data accumulates at the PM when outermost zones are active at the Scheduler and decreases when reading blocks from the innermost zones. In this figure, $T_{\mathrm{Mux}}(Z_i)$ denotes the duration of a time that a zone is active. It is longer for the innermost zone due to their low transfer rate. In essence, FIXB employs memory to compensate for the slow zones using the transfer rate of the fastest zones, harnessing the average disk transfer rate.

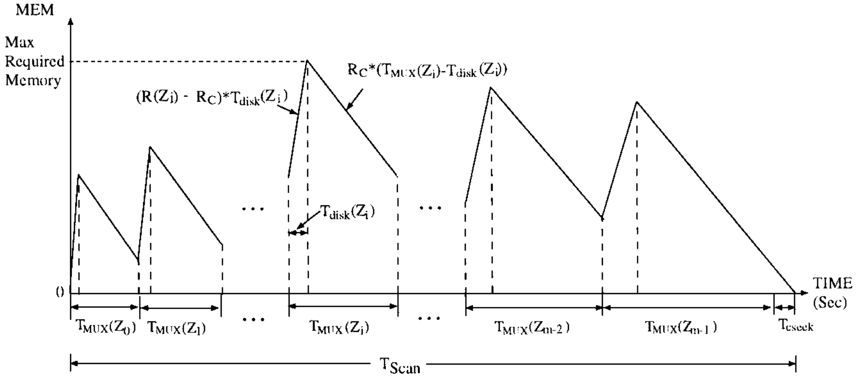


Figure 5. Memory requirement with FIXB.

If X_0 is assigned to a zone other than the outermost one (say Z_{X_0}) then its display may not start at the end of the time period that the system retrieves X_0 (i.e., $T_{\mathrm{MUX}}(Z_{X_0})$). This is because both the retrieval and display of data on behalf of a PM is synchronized relative to the transfer rate of the outermost zone to ensure that the amount of data produced during one sweep is equivalent to that consumed. If the display is not delayed, then the PM might run out of data and incur hiccups. By delaying the display at a PM, the system can avoid hiccups. In [19], we detail analytical models to compute the duration of a delay based on the identity of Z_{X_0}.

A drawback of recognizing a large number of zones is a higher startup latency. Mitra can reduce the number of zones by logically treating one or more adjacent zones as a single logical zone. This is achieved by overlaying a single EVEREST file system on these zones. Mitra assumes the transfer rate of the slowest participating zone as the transfer rate of the logical zone to guarantee hiccup-free displays.

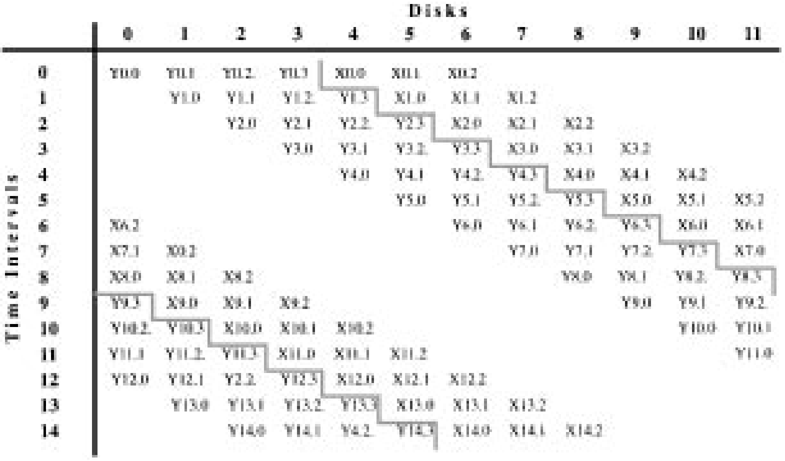
3.1.3. PM-driven scheduling. The duration of a time period might exceed the disk service time to retrieve $\mathcal{N}$ blocks on behalf of $\mathcal{N}$ active displays. This is because: 1) Mitra assumes the transfer rate of a logical zone to equal the transfer rate of the slowest participating physical zone, and 2) the $\mathcal{N}$ retrieved blocks might physically reside in the fastest participating zone (by luck). When this happens, the Scheduler may either (1) busy-wait until the end of the time period, (2) employ the idle slot for house keeping activities, i.e., migrate sections in support of the bounded list length property, or 3) proceed with the retrieval of blocks that should be retrieved during the next time period. The third approach minimizes the average startup latency of the system (as demonstrated in Section 4). However, it causes the Scheduler to produce data at a faster rate on behalf of an active PM. This motivates an implementation of a PM-driven scheduling paradigm where the Scheduler accepts skip messages from a PM when the PM starts to run out of memory.

With this paradigm, a PM maintains a data buffer with a low and a high water mark. These two water marks are a percentage of the total memory available to the PM. Once the high water mark is reached, the PM generates a skip message to inform the Scheduler that it should not produce data on behalf of this PM for a fixed number of time periods (say Y time periods). Y must be a multiple of the number of logical zones recognized on a disk (otherwise, Y is rounded to $\lfloor \frac{Y}{m} \rfloor$). This is due to the round-robin assignment of blocks of each object to the zones where a display cannot simply skip one zone when $m > 1$. The number of time periods is dependent on the amount of data that falls between the low and high water marks, i.e., the number of blocks cached. It must correspond to at least one sweep of the zones (T_{scan}) to enable the PM to issue a skip message. During the next Y time periods, the Scheduler produces no data on behalf of the PM while the display consumes data from buffers local to the PM. After Y time periods, the Scheduler starts to produce data for this PM.

The choice of a value for the low and high water marks at the PM are important. The difference between the total available memory and the high water mark should be at least one block due to possible race conditions attributed to networking delays between the PM and the Scheduler. For example, the Scheduler might produce a block for the PM at the same time that the PM is generating the skip message. Similarly, the low water mark should not be zero (its minimum value must be one block). This would eliminate the possibility of the PM running out of data (resulting in hiccups) due to networking delays.

3.2. *Staggered striping*

Staggered striping was originally presented in [3, 14]. This section describes its implementation in Mitra. With staggered striping, Mitra does not treat all the available disks (say D disks) as a single logical disk. Instead, it constructs *clusters* of disks, with each treated as a single logical disk. Assuming that the database consists of η media types, Mitra registers for each media type M_i: (1) the number of disks that constitute a cluster for this media type, termed $d(M_i)$, and (2) the block size for M_i, i.e., $B(M_i)$. (The tradeoff associated with alternative values for $d(M_i)$ and $B(M_i)$ is reported in Section 4.) Mitra constructs logical clusters (instead of physical ones) using a fixed stride value (k). This is achieved as follows. When loading an object (say X) of media type M_X, the first block of X (X_0) recognizes a cluster as consisting of $d(M_X)$ adjacent disks starting with an arbitrary disk (say disk$_a$). Mitra declusters X_0 into $d(M_X)$ fragments and assigns each fragments to a disk starting with disk$_a$: disk$_a$, disk$_{(a+1) \bmod D}$, $\ldots$, disk$_{(a+d(M_X)) \bmod D}$. For example, in figure 6, X_0 is declustered into three fragments $d(M_X) = 3$ and assigned to a logical cluster starting with disk 4. It places the remaining blocks of X such that the first disk that contains the first fragment of block X_j is shifted k disks to the right relative to that of block X_{j-1}. Thus, in our example, the placement of X_1 would start with disk$_b$ where $b = (a + k) \bmod D$. The placement of X_2 starts with disk$_{(b+k) \bmod D}$. In figure 6, $k = 1$. Thus, X_1 is declustered across disks 5, 6, and 7 while X_2 is declustered across disks 6, 7, and 8. With m zones per disk, the assignment of blocks to the zones of clusters continues to follow a round-robin assignment. For example, if X_0 is assigned to zone Z_i of disks a to $(a + d(M_X)) \bmod D$, X_1 is assigned to zone $Z_{(i+1) \bmod m}$ of disks b to $(b + k) \bmod D$. This process repeats until all blocks of X are assigned to disks and zones. One EVEREST file contains all fragments of X assigned to zone i of disk j. Thus,



Time Intervals	Disks											
	0	1	2	3	4	5	6	7	8	9	10	11
0	Y0.0	Y0.1	Y0.2	Y0.3	X0.0	X0.1	X0.2					
1		Y1.0	Y1.1	Y1.2	Y1.3	X1.0	X1.1	X1.2				
2			Y2.0	Y2.1	Y2.2	Y2.3	X2.0	X2.1	X2.2			
3				Y3.0	Y3.1	Y3.2	Y3.3	X3.0	X3.1	X3.2		
4					Y4.0	Y4.1	Y4.2	Y4.3	X4.0	X4.1	X4.2	
5						Y5.0	Y5.1	Y5.2	Y5.3	X5.0	X5.1	X5.2
6	X6.2						Y6.0	Y6.1	Y6.2	Y6.3	X6.0	X6.1
7	X7.1	X7.2						Y7.0	Y7.1	Y7.2	Y7.3	X7.0
8	X8.0	X8.1	X8.2						Y8.0	Y8.1	Y8.2	Y8.3
9	Y9.3	X9.0	X9.1	X9.2						Y9.0	Y9.1	Y9.2
10	Y10.2	Y10.3	X10.0	X10.1	X10.2						Y10.0	Y10.1
11	Y11.1	Y11.2	Y11.3	X11.0	X11.1	X11.2						Y11.0
12	Y12.0	Y12.1	Y2.2	Y12.3	X12.0	X12.1	X12.2					
13		Y13.0	Y13.1	Y13.2	Y13.3	X13.0	X13.1	X13.2				
14			Y14.0	Y14.1	Y4.2	Y14.3	X14.0	X14.1	X14.2			

Figure 6. Staggered striping for two media types.

a total of $D \times m$ files might represent object X. Once object X is loaded, Mitra registers with the catalog the following information: (1) the disk and zone that the assignment of X_0 started with, (2) X's media type, and (3) the identity of each file that contains different fragments of X.

While the value of $d(M_i)$ might differ for the alternative media types, k is a constant for all media types. For example, in figure 6, the media type of object X requires the bandwidth of three disks while that of Y requires four disks. However, the value of $k = 1$ for both objects.

To display an object X, the Scheduler uses the catalog to determine: (1) X's media type, i.e., the value of $d(M_X)$ for this object, (2) the disk that contains the first fragment of X_0 (say disk$_a$), and (3) the zone that contains X_0 (say Z_{X_0}). Once the active zone equals Z_{X_0} and $d(M_X)$ disks starting with disk$_a$ (i.e., disk$_a$, disk$_{(a+1) \bmod D}$, ..., disk$_{(a+d(M_X)) \bmod D}$) have sufficient bandwidth to retrieve the fragments of X_0, the Scheduler initiates the retrieval of X_0. During the next time period, this display shifts k disks to the right and the next active zone to retrieve X_1. This process repeats until all blocks of X have been retrieved and transmitted to the PM.

4. Performance evaluation

This section presents performance numbers that demonstrate the scalability characteristics of Mitra. We start with an overview of the experimental design employed for this evaluation. Next, we focus on a single disk configuration of Mitra to demonstrate the tradeoff associated with its alternative optimization techniques. Finally, we present the performance of Mitra as a function of the number of disks in the system and their logical organization as clusters. In all experiments, the entire system was dedicated to Mitra with no other users accessing the workstations.

4.1. Experimental design

A problem when designing this evaluation study was the number of variables that could be manipulated: block size, number of groups with GSS, mix of media types, mix of requests, the number of participating disks, the number of disks that constitute a cluster per media type, the bandwidth of each disk as a function of the number of participating disks, closed versus open evaluation, the role of the tertiary storage device, the size of database, frequency of access to objects that constitute the database, etc. We spent weeks analyzing alternative ways of conducting this study. It was obvious that we had to reduce the number of manipulated parameters to obtain meaningful results. As a starting point, we decided to: (1) ignore the role of tertiary storage device and focus on the performance of Mitra during a steady state where all referenced objects are disk resident, and (2) focus on a single media type. Moreover, we partitioned this study into two parts. While the first focused on the performance of a single disk and the implementation techniques that enhance its performance, the second focuses on the scalability characteristics of Mitra as a function of additional disks.

The target database and its workload were based on a WWW page that ranks the top fifty songs every week[7]. We chose the top 22 songs of January 1995 to construct both the benchmark database and its workload. (We could not use all fifty because the total size of the top 22 audio clips exhausted the storage capacity of one disk Mitra configuration.) Figure 7(a) and (b) shows the frequency of access to the clips and the size of each clip in seconds, respectively. The size of the database was fixed for all experiments.

We employed a closed evaluation model with a zero think time. With this model, a workload generator process is aware of the number of simultaneous displays supported by a configuration of Mitra (say $\mathcal{N}$). It dispatches $\mathcal{N}$ requests for object displays to Mitra. (Two or more requests may reference the same object, see below.) As soon as Mitra is done with the display of a request, the workload generator issues another request to the Scheduler (zero think time). The distribution of request references to clips is based on figure 7(a). This is as follows. We normalized the number of votes to the 22 clips as a function of the total number of vote for these objects. The workload generator employs this distribution to construct a queue of requests that reference the 22 clips. This queue of requests is randomized to result in a non-deterministic reference pattern. However, it might be the case that two or more

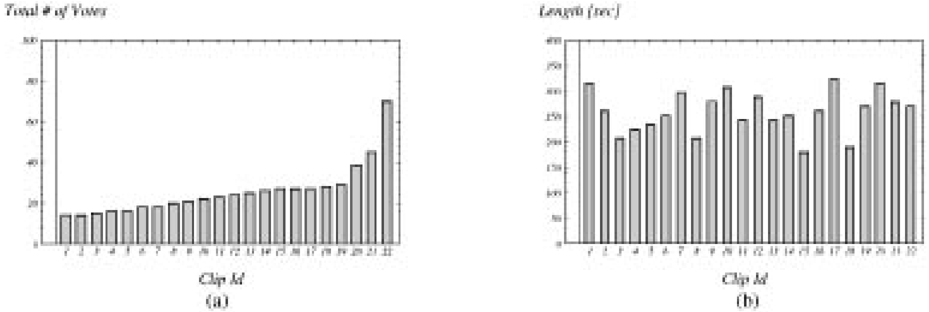


Figure 7. Characteristics of the CD audio clips. (a) Number of votes for each clips. (b) Length [in seconds] of each clip.

Table 3. Fixed parameters.

Seagate ST31200W	
Capacity	1.0 gigabyte
Revolutions per minute	5400
Maximum seek time	21.2 millisecond
Maximum rotational latency	11.1 millisecond
Number of zones	23 (see figure 5)
Database Characteristics *CD Quality Audio*	
Sampling rate	44,100 per second
Resolution	16 bits
Channels	2 (stereo)
Bandwidth requirement	1.3458 Mbps

requests reference the same clip (e.g., the popular clip) at the same time. In all experiments Mitra was configured NOT to multiplex a single stream to service these requests.

This experimental design consists of three states: warmup, steady state, and shutdown. During the system warmup (shutdown), Mitra starts to become fully utilized (idle). In our experiments, we focused on the performance of Mitra during a steady state by collecting no statistics during both system warmup and shutdown.

4.2. One disk configuration

We analyzed the performance of Mitra with a single disk to observe the impact of: 1) alternative mode of operation with the PM-driven scheduling paradigm, 2) block size, 3) different number of groups with GSS, and 4) multiple zones. In the first experiment, we configured the system with 384 KByte block size, $g = 1$, a single zone, with the low and high water marks set to 1 and 2 respectively. In theory, the number of guaranteed simultaneous displays supported by our target disk is 12. This is computed based on the transfer rate of the slowest zone, i.e., 18 Mbps, to capture the worst case scenario where all blocks retrieved during a time period reside in this zone. Mitra realized these theoretical expectations successfully. However, during a time period, the referenced blocks might be scattered across the disk surface, causing the system to observe the average disk transfer rate (26 Mbps). This results in a number of idle slots per time period. The PM-driven scheduling approach to proceed with the retrieval of blocks for the next time period (see Section 3.1.3) reduces the average latency when compared with busy waiting (0.3 seconds compared with 2.4 seconds). This paradigm enhances the probability of a new request locating an idle slot during the current time period.

In the second experiment, we changed the block size from 32 KByte to 64, 128, and 256 KByte. (The remaining parameters are unchanged as compared with the first experiment.) As the block size increases, Mitra supports a higher number of simultaneous displays (6, 8, 10, and 12 displays, respectively). The maximum number of simultaneous displays supported by the available disk bandwidth is 13 and can be realized with a block size of 625 KByte[8]. The explanation for this is as follows. With magnetic disks, the block size impacts the percentage of wasted disk bandwidth attributed to seek and rotational delays. As the block size increases, the impact of these delays becomes less significant, allowing the disk to support a higher number of simultaneous displays [9].

The number of groups (g) with GSS impacts the seek times incurred by the disk when retrieving blocks during a time period. In general, small values of g minimize the seek time. The number of groups (g) has an impact with small block sizes where the seek time is significant. This impact becomes negligible with large block sizes. For example, with a 64 KByte block size, Mitra supports 6 displays with six groups, 7 displays with three groups, and 8 displays with one group. However, with a 384 KByte block, Mitra supports 11 displays with eleven groups, and 12 displays with one group. This block size is large enough to render the seek time insignificant when compared with the transfer time of a block.

In a final experiment, EVEREST was configured to recognize all the 23 zones of the disk. The block size was 539 KByte to guarantee a continuous display with FIXB. In this case, Mitra can store only twelve clips (instead of 22) on the disk because once the storage capacity of the smallest zone is exhausted, no additional clips can be stored (due to a round-robin assignment of blocks to zones). With this configuration, Mitra supports 17 displays with an average startup latency of 35.9 seconds. The higher number of simultaneous displays (as compared to 12 in the previous experiments) is due to the design of FIXB that enables Mitra to harness the average disk transfer rate. The higher startup latency is because a display must wait until the zone containing its first block is activated. The number of logical zones recognized by Mitra is a tradeoff between the number of displays supported by the system, the average startup latency and the percentage of wasted disk space. We now report on several experiments that demonstrate this tradeoff. In the first experiment, we configured EVEREST to recognize two logical zones. The first logical zone consists of zones Z_0 to Z_{11} while the second consists of the remaining physical zones. In this case, Mitra can store 15 clips on the disk. With this configuration, while the number of simultaneous displays is reduced to 14, the average startup latency is reduced to 0.22 seconds. In a second experiment, we configured EVEREST to recognize one logical zones consisting of only the nine outermost zones. With this configuration, Mitra can store twelve clips on the disk because EVEREST has eliminated the storage capacity of the 14 innermost zones. This increases the transfer rate, allowing Mitra to support 19 displays with an average startup latency of 2 seconds. The higher startup latency is due to a longer duration of a time period. In [19], we detail a planner that determines system parameters to satisfy the performance objectives of an application (it desired throughput and maximum startup latency tolerated by its clients).

4.3. Multi-disk configuration

In these experiments, the following system parameters are fixed: block size is 384 KByte, GSS is configured with a single group ($g = 1$), and a single logical zone spans all 23 physical zones of each disk. We analyzed the performance of Mitra as a function of additional disks by varying D from 1 to 2, 4, 8, and 12. For each configuration, we analyzed the

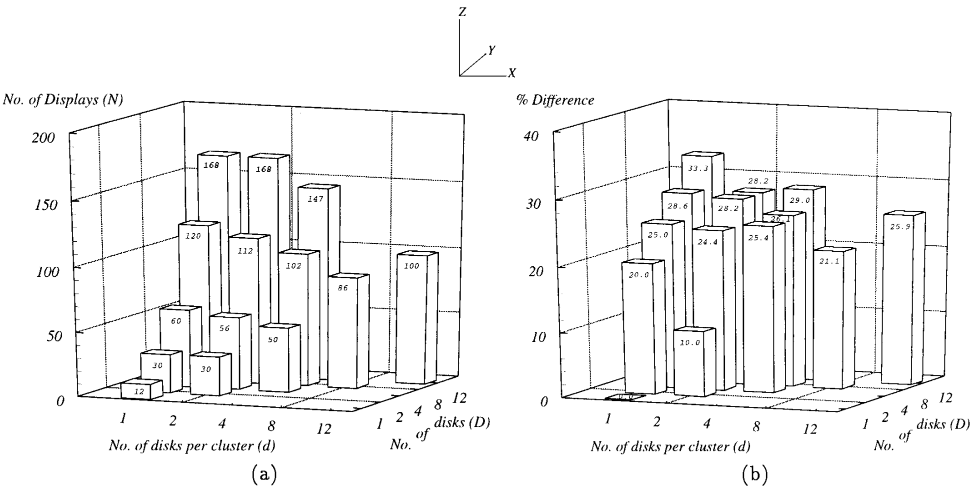


Figure 8. Performance of Mitra as a function of D and d ($k = d$). (a) Number of simultaneous displays. (b) Percentage difference between theoretical expectations and obtained results from Mitra.

performance of Mitra as a function of the number of disks that constitute a cluster (i.e., d). In all experiments, the stride (k) equals to d. For example, with a 12 disk configuration ($D = 12$), a cluster may consist of two disks ($d = 2$). With this configuration, stride would also equal to two ($k = 2$). Obviously, the choice of d and k has a significant impact on the obtained results. We analyze the performance of Mitra for those values of d and k that are reasonable[9]. For example, with $D = 12$, it would be unreasonable to configure Mitra with $d = k = 8$ because it would force the bandwidth of four disks to sit idle because the database consists of a single media type. With $d = k = 8$, the performance of Mitra with $D = 12$ would be reduced to that with $D = 8$. Figure 8(a) presents the number of simultaneous displays supported by Mitra as a function of D and d. In this figure, the number of disks available to Mitra is varied on the y-axis, the number of disks that constitute a cluster is varied on the x-axis, and the throughput of the system is reported on the z-axis.

As the number of disks in the system (D) increases from 1 to 12 with $d = k = 1$, the throughput of the system increases super linearly (the throughput of Mitra with $D = 12$ is fourteen times higher than that with $D = 1$). This is because the average transfer rate of each disk increases as a function of D. The explanation for this is as follows. In this experiment, the size of the database is fixed and the EVEREST file system organizes files on a disk starting with the outermost zone, i.e., fastest zone. The amount of data assigned to each disk shrinks as D increases. With $D = 1$, the innermost zone of the disk contains data, while with $D = 12$, only the three outermost zones contain data. The average transfer rate of the three outermost zones is higher than the average transfer rate of all 23 zones of a disk (see figure 5).

In figure 8(b), for a given hardware platform (fixed D), the throughput of Mitra drops as d increases. For example with $D = 12$, Mitra's throughput drops from 168 streams to 100 as d increases from 1 to 12. This is because the percentage of wasted disk bandwidth increases as d increases in value [14]. To observe this, note that both the maximum seek time and rotational latency are fixed. Moreover, they waste disk bandwidth. The percentage of wasted disk bandwidth is a function of these two values along with the amount of data read from each disk drive per time period. As d increases in value, the amount of data retrieved from each disk decreases because a block is declustered across a larger number of disks. This wastes a higher percentage of disk bandwidth, resulting in a lower throughput.

For each choice of D, we located the slowest participating zone of the disks that contains data. This zone is the same for all disks due to the round-robin assignment of blocks of each object to disks. We computed expected performance of Mitra as a function of this zone's transfer rate for each configuration (using the analytical models of [14, 15]). Next, we examined how closely Mitra approximates these theoretical expectations. Figure 8(b) presents the percentage difference between the measured results and theoretical expectations. Each value of this figure is computed based on: $1 - \frac{\text{Measured}}{\text{Theory}}$. With $D = 1$, the system approximates the theoretical expectation with 100% accuracy. With $D > 1$, Mitra's performance is anywhere from 10% to 35% lower than its theoretical expectations. Part of this is due to loss of network packets using UDP and their retransmission with HP-NOSE. However, there are other factors (e.g., SCSI-2 bus, software overhead, system bus arbitration, HP-UX scheduling of processes, etc.,) that might contribute to this difference. These delays are expected with a software based system (based on previous experience with Gamma [7] and Omega [11]) because the system does not have complete control on the underlying hardware.

A limitation associated with values of d smaller than D is that the placement of data is constrained with staggered striping. This results in a higher average startup latency (see figure 9(a)). In addition, it increases the amount of memory required at each PM even with the PM-driven scheduling paradigm (see figure 9(b)). Consider each observation in turn. The average startup latency is higher because a display must wait until the cluster containing its first subobject has sufficient bandwidth to retrieve its referenced block. Similarly, each PM requires a larger amount of memory because the Scheduler cannot simply skip one time

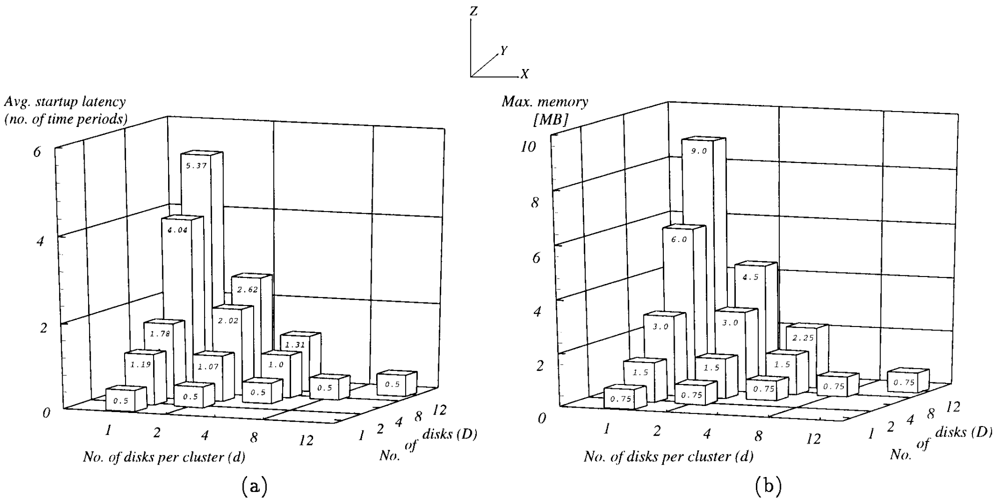


Figure 9. Startup latency and memory requirement of a PM with Mitra. (a) Average startup latency. (b) Maximum amount of memory required at a PM.

period on its behalf. Its next block resides on the cluster adjacent to the currently active cluster. Assuming the system consists of $\mathcal{C}$ clusters, a PM must cache enough data so that the Scheduler skips multiples of $\mathcal{C}$ time periods on behalf of this PM.

5. Conclusion and future research directions

Mitra is a scalable storage manager that support the display a mix of continuous media data types. Its primary contribution is a demonstration of several design concepts and how they are glued together to attain high performance. Its performance demonstrates that an implementation can approximate its theoretical expectations.

As part of our future research direction, we are extending Mitra in several novel directions. First, we have introduced techniques to support on-line re-organization of data when new disks are added to a system that has been in operation for a while [20]. These technique modify the placement of data to incorporate new disks (both their storage and bandwidth) without interrupting service. Second, we are investigating several designs based on request-migration and object replication to minimize the startup latency of the system [14]. Third, we are evaluating techniques that speedup the rate of display to support VCR functionalities such as fast-forward and fast-rewind. These techniques are tightly tied to those of the second objective that minimize the startup latency of a display. Finally, we are investigating distributed buffer pool management technique to facilitate sharing of a single stream among multiple PMs that are displaying the same presentation. The buffer pool is distributed across the available DMs. However, its content is controlled by the Scheduler.

Appendix A: Admission control with GSS

This appendix details the implementation of the Scheduler's admission policy with GSS. A building component is a function, termed seek(# cyl), that estimates the disk seek time. Its input is the number of cylinders traversed by the seek operation. Its output is an estimate of the time required to perform the seek operation using the models of [16]. Assuming CYL cylinders for the disk and n displays assigned to a group G_i, we assume that the n blocks are $\frac{\text{CYL}}{n}$ cylinders apart.

The Scheduler maintains the amount of idle time left for each group G_i. With a new request for object X, the scheduler retrieves from the catalog the record corresponding to X to determine its media type, M_X. Next, it retrieves from the catalog the record corresponding to media type M_X to determine $\mathcal{B}(M_X)$. Starting with the current group G_i, the Scheduler compares the idle time of G_i with the disk service time to retrieve a block of size $\mathcal{B}(M_X)$. The disk service time with G_i is:

$$S_{\text{disk}}(G_i) = \frac{\mathcal{B}(M_X)}{\mathcal{R}_D} + \text{max rotational latency} + \text{seek(CYL)}$$

It assumes the maximum seek time (i.e., seek(CYL)) because the blocks to be retrieved during G_i have already been scheduled and the new request cannot benefit from the scan policy. Assuming that G_i is servicing $n - 1$ requests and its idle time can accommodate

$S_{\text{disk}}(G_i)$, its idle time is reduced by $S_{\text{disk}}(G_i)$. Prior to initiating the retrieval of blocks that belong to group G_{i+1}, the scheduler adjusts the idle time of group G_i to reflect that the active requests can benefit from the scan policy. Thus, the idle time of G_i is adjusted as follows:

$$\begin{aligned}\text{idle}(G_i) = \text{idle}(G_i) - \left[\text{seek}(\text{CYL}) + (n-1) \times \text{seek}\left(\frac{\text{CYL}}{n-1}\right)\right] \\ + \left[n \times \text{seek}\left(\frac{\text{CYL}}{n}\right)\right]\end{aligned}$$

The subtracted portion reflects the maximum seek time of the request that was just scheduled and the seek time of $n-1$ other active requests. The added portion reflects the n seeks incurred during the next time period by this group with each $\frac{\text{CYL}}{n}$ cylinders apart.

If current group G_i has insufficient idle time, the Scheduler proceeds to check the idle time of other groups G_j where $j = (i+1) \bmod g$, $0 < j < g$ and $j \neq i$. Assuming that G_j is servicing $n-1$ active requests, the disk service time with G_j is:

$$\begin{aligned}S_{\text{disk}}(G_j) = \frac{\mathcal{B}(M_X)}{\mathcal{R}_D} + \text{ max rotational latency} \\ + \left[n \times \text{seek}\left(\frac{\text{CYL}}{n}\right)\right] - \left[(n-1) \times \text{seek}\left(\frac{\text{CYL}}{n-1}\right)\right]\end{aligned}$$

If the idle time of G_j is greater than $S_{\text{disk}}(G_j)$, then the new request is assigned to G_j and its idle time is subtracted by $S_{\text{disk}}(G_j)$.

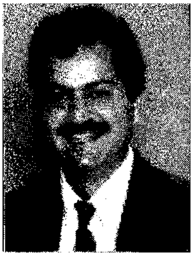
Notes

1. We do not report on commercial systems due to lack of their implementation detail, see [27] for an overview of these systems.
2. With the buffered interface of the HP-UX file system, one might read and write a single byte. This functionality is supported by a buffer pool manager that translates this byte read/write to a $\frac{1}{2}$ KByte read/write against the physical device.
3. The size of a page has no impact on the granularity at which a process might read a section. This is detailed below.
4. To simplify the discussion, assume that the total number of pages is a power of ω. The general case can be handled similarly and is described below.
5. A lazy variant of this scheme would allow these lists to grow longer and do compaction upon demand, i.e., when large contiguous pages are required. This would be complicated as a variety of choices might exist when merging pages. This would require the system to employ heuristic techniques to guide the search space of this merging process. However, to simplify the description we focus on an implementation that observes the invariant described above.
6. We intend to implement VARB in the near future.
7. This web site is maintained by Daniel Tobias (http://www.softdisk.com/comp/hits/). The ranking of the clips is determined through voting by the Internet community, via E-mail.
8. Thirteen is computed based on the bandwidth of the innermost zone, consumption rate of CD-quality audio, and maximum seek and rotational latency times.
9. However, the results are presented such that one can estimate the performance of the system with unreasonable choice of d and k values.

References

1. D.P. Anderson, Y. Osawa, and R. Govindan, “Real-time disk storage and retrieval of digital audio/video data,” IEEE Transactions on Computer Systems, 1992.
2. M.M. Astrahan, M.W. Blasgen, D.D. Chamberlin, and et al, “System R: Relational approach to database management,” ACM Transactions on Database Systems, Vol. 1, No. 2, pp. 97–137, June 1976.
3. S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, “Staggered striping in multimedia information systems,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1994.
4. M. Carey, L. Haas, and M. Livny, “Tapes hold data, too: Challenges of tuples on tertiary storage,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 413–417, 1993.
5. H.T. Chou, D.J. DeWitt, R. Katz, and T. Klug, “Design and implementation of the Wisconsin storage system,” Software Practices and Experience, Vol. 15, No. 10, 1985.
6. P.J. Denning, “Working sets past and present,” IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, pp. 64–84, Jan. 1980.
7. D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen, “The gamma database machine project,” IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 2, March 1990.
8. J. Gemmell, H. Beaton, and S. Christodoulakis, “Delay sensitive multimedia on disks,” IEEE Multimedia, 1994.
9. D.J. Gemmell, H. Vin, D.D. Kandlur, P. Rangan, and L. Rowe, “Multimedia storage servers: A tutorial,” IEEE Computer, Vol. 28, No. 5, pp. 40–51, May 1995.
10. S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi, “Object placement in parallel hypermedia systems,” in Proceedings of the International Conference on Very Large Databases, 1991.
11. S. Ghandeharizadeh, V. Choi, C. Ker, and K. Lin, “Omega: A parallel object-based system,” in Proceedings of the International Conference on Parallel and Distributed Information Systems, Dec. 1992.
12. S. Ghandeharizadeh and L. Ramos, “Continuous retrieval of multimedia data using parallelism,” IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 4, Aug. 1993.
13. S. Ghandeharizadeh and D. Ierardi, “Management of disk space with REBATE,” Proceedings of the Third International Conference on Information and Knowledge Management (CIKM), Nov. 1994.
14. S. Ghandeharizadeh and S.H. Kim, “Striping in multi-disk video servers,” in SPIE International Symposium on Photonics Technologies and Systems for Voice, Video, and Data Communications, Philadelphia, Pennsylvania, Oct. 1995.
15. S. Ghandeharizadeh, S.H. Kim, and C. Shahabi, “On configuring a single disk continuous media server,” in Proceedings of the ACM SIGMETRICS, 1995.
16. S. Ghandeharizadh, J. Stone, and R. Zimmermann, “Techniques to quantify SCSI-2 disk subsystem specifications for multimedia,” Technical Report USC-CS-TR95-610, USC, 1995.
17. S. Ghandeharizadeh, D. Ierardi, D.H. Kim, and R. Zimmermann, “Placement of data in multi-zone disk drives,” in Second International Baltic Workshop on Databases and Information Systems, June 1996.
18. S. Ghandeharizadeh, D. Ierardi, and R. Zimmermann, “An on-line algorithm to optimize file layout in a dynamic environment,” Information Processing Letters, No. 57, pp. 75–81, 1996.
19. S. Ghandeharizadeh, S.H. Kim, C. Shahabi, and R. Zimmermann, “Placement of Continuous Media in Multi-Zone Disks,” Multimedia Information Storage and Management, S. Chung (Ed.), Kluwer, 1996.
20. S. Ghandeharizadeh and D. Kim, “On-line re-organization of data in continuous media servers,” Multimedia Tools and Applications, Jan. 1997.
21. J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, pp. 682–684, 1993.
22. Hewlett-Packard Co. How HP-UX Works: Concepts for the System Administrator, 1991.
23. J. Hsieh, M. Lin, J. Liu, D. Du, and T. Ruwart, “Performance of a mass storage system for video-on-demand,” Journal of Parallel and Distributed Computing, Vol. 30, pp. 147–167, 1995.
24. K.C. Knowlton, “A fast storage allocator,” Communications of the ACM, Vol. 8, No. 10, pp. 623–625, Oct. 1965.
25. H.R. Lewis and L. Denenberg, Data Structures & Their Algorithms, Harper Collins, Chapter 10, pp. 367–372, 1991.

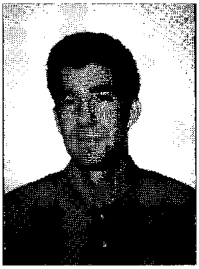
26. P. Lougher and D. Shepherd, "The design and implementation of a continuous media storage server, network and operating system support for digital audio and video," in Proceedings of the 3rd International Workshop, La Jolla CA, Springer Verlag, pp. 69–80, 1992.
27. K. Natarajan, "Video servers take root," in IEEE Spectrum, pp. 66–69, April 1995.
28. B. Özden, R. Rastogi, and A. Silberschatz, "Fellini-a file system for continuous media," Technical Report 113880–941028–30, AT&T Bell Laboratories, Murray Hill, 1994.
29. B. Özden, R. Rastogi, and A. Silberschatz, "The storage and retrieval of continuous media data," Multimedia Database Systems, V.S. Subrahmanian and S. Jojodia (Eds.), Springer, 1996.
30. R. Rooholamini and V. Cherassky, "ATM based multimedia servers," IEEE Multimedia, Vol. 2, No. 1, pp. 39–53, 1995.
31. C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," IEEE Computer, March 1994.
32. F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID-A disk array management system for video files," in First ACM Conference on Multimedia, Aug. 1993.
33. P.S. Yu, M.-S. Chen, and D.D. Kandlur, "Grouped sweeping scheduling for DASD-based multimedia storage management," Multimedia Systems, Vol. 1, No. 1, pp. 99–109, Jan. 1993.



Shahram Ghandeharizadeh received his Ph.D. degree in computer science from the University of Wisconsin, Madison, in 1990. In 1992, he received the National Science Foundation Young Investigator's Award for his research on the physical design of parallel database systems.

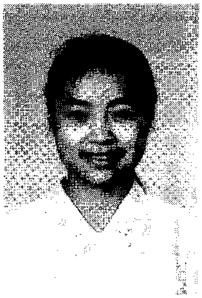
In 1995, he received an award from the School of Engineering at USC in recognition of his research activities. His primary research interests include design and implementation of multimedia storage managers, parallel database systems, and active databases.

His activities are supported by several grants from the National Science Foundation, Department of Defense, and Hewlett-Packard.

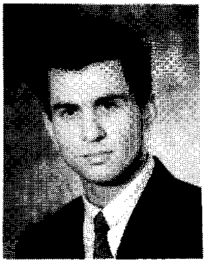


Roger Zimmermann received his first undergraduate degree, Informatik Ingenieur HTL, from the HTL Brugg-Windisch engineering college, Switzerland, in 1986. Afterwards, he worked as a software engineer for Zühlke Engineering AG in Switzerland until 1991. He received his B.S. and M.S. degrees in computer science from the University of Southern California, Los Angeles, in 1992 and 1994, respectively. He is currently pursuing a Ph.D. degree in computer science also at the University of Southern California.

His main research interests are in the area of novel database architectures, specifically object-oriented and multimedia databases. His current work focuses on high-performance continuous-media servers, specifically, on how heterogeneous storage subsystems consisting of different magnetic disk drive types can be effectively utilized.

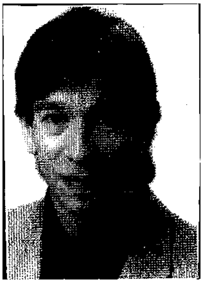


Weifeng Shi received the B.S. and M.S. degrees in Computer Science from Peking University, Beijing, China, in 1991 and 1994 respectively. She is currently working towards the Ph.D. degree in computer science at the University of Southern California. Her research interests include design and analysis of continuous data sharing schemes, multimedia storage management and interactive multimedia.



Reza Rejaie is a research assistant with the Information Science Institute at the University of Southern California. He is currently pursuing his doctoral research in the area of multimedia communication networks. His research interests includes congestion control mechanism for real time data streams and continuous media servers.

He received his B.S. degree in Electrical Engineering from Sharif University of Technology at Tehran, Iran in 1991 and his M.S. degree in Computer Science from the University of Southern California in 1996.



Doug Ierardi received the B.A. degree from Yale University, and the M.S. and Ph.D. degrees from Cornell University in Computer Science in 1987 and 1989, respectively. During the 1989–90 academic year, he was a Research Associate at Princeton University, in the Department of Computer Science and at the NSF Center for Discrete Mathematics and Theoretical Computer Science. From 1990 to 1996, he was an Assistant Professor in the Computer Science, Department of USC, where he continues to hold an adjunct appointment. He is currently a principal on the engineering staff at The Walt Disney Company and a Visiting Lecturer at the University of

California, Los Angeles. His research interests include applied computational geometry and scheduling for multimedia data.

Ta-wei Li received the B.S. degree in Computer Science from the University of Southern California in 1996. He is currently working on the master degree in computer science in the University of Southern California. His research interestes are scheduling of multimedia system and delivery of multimedia data over network.