The Evaluation of Video Layout Strategies for a High-Performance Storage Server

Kimberly Keeton
Department of Electrical Engineering and Computer Science
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720

Technical Report: UCB//CSD-95-889

The Evaluation of Video Layout Strategies for a High-Performance Storage Server

Abstract

We propose a systems approach to providing video service that integrates the multi-resolution data generated by scalable compression algorithms with the high-bandwidth, high-capacity storage provided by disk arrays. We introduce two layout strategies for storing multi-resolution video data on magnetic disk arrays, which vary in the degrees of parallelism and concurrency they use to satisfy requests. We also present the event-driven simulator that we used to evaluate these layout strategies. Our simulation results show that striping video data over disks in an array can provide up to a two-fold increase in the number of viewers supported. In addition, the storage of multiple video resolutions allows a video file server to satisfy considerably more user requests than a server that stores a single resolution of video data.

Contents

1.0 Introduction

Recent advances in workstation architecture, high-speed networking and storage technologies have enabled new kinds of multimedia systems. One such system is a multimedia server that supports the playback of stored video to client display devices. Video storage servers must satisfy the requirements of individual video streams as well as system-wide requirements. Because video files are considerably larger than traditional files, they require significant storage capacity. Real-time video playback imposes strict delay and delay variance requirements on the retrieval of video files. From a system-wide perspective, video servers must also maximize the number of concurrent requests that can be satisfied, without violating any of the clients' performance requirements.

Video servers must also be able to support heterogeneous clients that request vastly different quality of service (QoS) parameters, such as display size, resolution, and frame rate. Video storage systems typically store one representation of each video sequence, which is used to satisfy all client requests. This representation may contain more information than is necessitated by the requested QoS, wasting server resources. Moreover, client resources are also wasted in dynamically adjusting the QoS parameters (in addition to performing decompression) before displaying the video sequence.

One possible solution to the problem of managing different QoSs is to store multiple representations of video on the server. Each request can then be satisfied with a representation that closely matches the requested QoS, and requires only decompression for playback. The storage of multiple representations of data may also be used to accommodate as many playback requests as possible; in case of overload, the server may switch the representations of some streams to use those that place less demand on the server, the network, or both.

The high concurrency of access and high data throughput rates required by video retrieval make single disk systems inadequate for video service. We address these issues by taking advantage of the high concurrency, bandwidth, and capacity provided by disk arrays such as the RAID (Redundant Arrays of Inexpensive Disks) prototypes developed at Berkeley [Che91,Che93].

In this report, we present an integrated systems approach to video service that investigates the storage of multiple resolutions of video data on disk arrays; this work was introduced in [Kee93]. In Section 2.0, we survey the existing approaches for video storage and retrieval. In Section 3.0, we discuss scalable compression algorithms, which generate video streams with multiple resolutions and rates. Section 4.0 presents the structure of video data, as well as a method of scheduling data retrieval. Section 5.0 compares several storage alternatives, including disk arrays, and introduces two data layout strategies, which use information about the video data structure and disk array parameters to store video data for efficient playback. We describe our simulator in Section 6.0, and present our simulation results in Section 7.0.

2.0 Related Work

Video playback generally requires periodically scheduled sequential reads under strictly limited delays. Existing file system data layout strategies support this access pattern to varying degrees. Traditional file systems, such as the BSD Fast File System [McK84] and Sprite LFS [Ros91] use layout strategies that do not take into account the video data structure. Because a given playback request may be stored in a non-contiguous manner, violations of performance requirements may occur.

Several file systems address the requirements of multimedia data, such as continuous storage and sequential retrieval of media. In particular, Rangan uses a constrained allocation policy for continuous media data blocks, to guarantee continuous access [Ran91]. Lougher stores data in an append-only log to exploit the temporal locality: data blocks recorded at the same time are typically replayed at the same time [Lou93]. To support multiple concurrent requests, both approaches develop admission control algorithms for determining whether a new user request can be accepted without violating existing real-time constraints. These video storage systems primarily focus on single disk systems or very small disk arrays, and do not address the issue of compression to support client heterogeneity.

Very few attempts have been made to take a systems view towards integrating image coding and data layout for video storage systems. One such approach is taken by Chiueh and Katz, who propose a multi-resolution video coding scheme based on Gaussian and Laplacian Pyramids for storing data on disk arrays [Chi93]. We now explore the use of disk arrays to store multi-resolution video generated by subband coding algorithms.

3.0 Scalable Compression Algorithms

To address the issue of client heterogeneity, we explore the use of scalable compression in order to support video data stored in multiple resolutions.

3.1 Motivation for Scalable Compression

Scalable compression algorithms play an important role in current and future video playback environments. A typical video storage system generally stores one video resolution per sequence, which it uses to satisfy all client requests. This single version must contain the highest (full) resolution data. However, client display characteristics, and hence client playback requirements, vary widely. For instance, workstations utilize large displays capable of holding multiple resizable windows, while portable computers typically use much smaller physical displays, capable of supporting smaller windows. Because the server has only a single version of the video, full-resolution data must be used to satisfy each window's video playback request, regardless of requested QoS parameters such as the window size and resolution. This places undue burden on the server I/O system, the network, and the client I/O system. In addition, the client must dynamically adjust the quality of received video (in addition to decompressing the compressed stream).

Scalable compression algorithms address these issues. This class of algorithms produces one compressed full-resolution video bit stream. It uses a hierarchical format where combinations of various components of the hierarchy correspond to different resolutions or rates of the same video sequence. Servers that use scalable compression can use different subsets to satisfy clients requesting different QoSs. For example, instead of receiving two full-resolution bit streams, a workstation with two small windows would receive two subsets of a lower resolution. Because these combinations have a lower bit rate than the full-resolution version, this relieves the load on the server and client I/O systems and network resources. In addition, the computational burden on the client is reduced because the video stream closely matches the requested QoS.

3.2 Subband Video Coding

One approach to scalable spatial and temporal compression uses subband coding techniques [Woo91]. Subband coding schemes typically use multi-stage algorithms, where each stage applies quadrature mirror filtering (QMF) to split the bandwidth of the input spectrum into low-band and high-band halves. More specifically, a single stage of the algorithm performs high- and low-pass filtering, followed by down-sampling, in the target dimension. Spatial (or intraframe) compression is performed by applying two stages of the algorithm, one in the x-direction and one in the y-direction, as shown in Figure 1. The corresponding decompression scheme is shown in Figure 1 [Cha93a]. Temporal (or interframe) compression can be achieved by performing the filtering and down-sampling or up-sampling among successive frames.

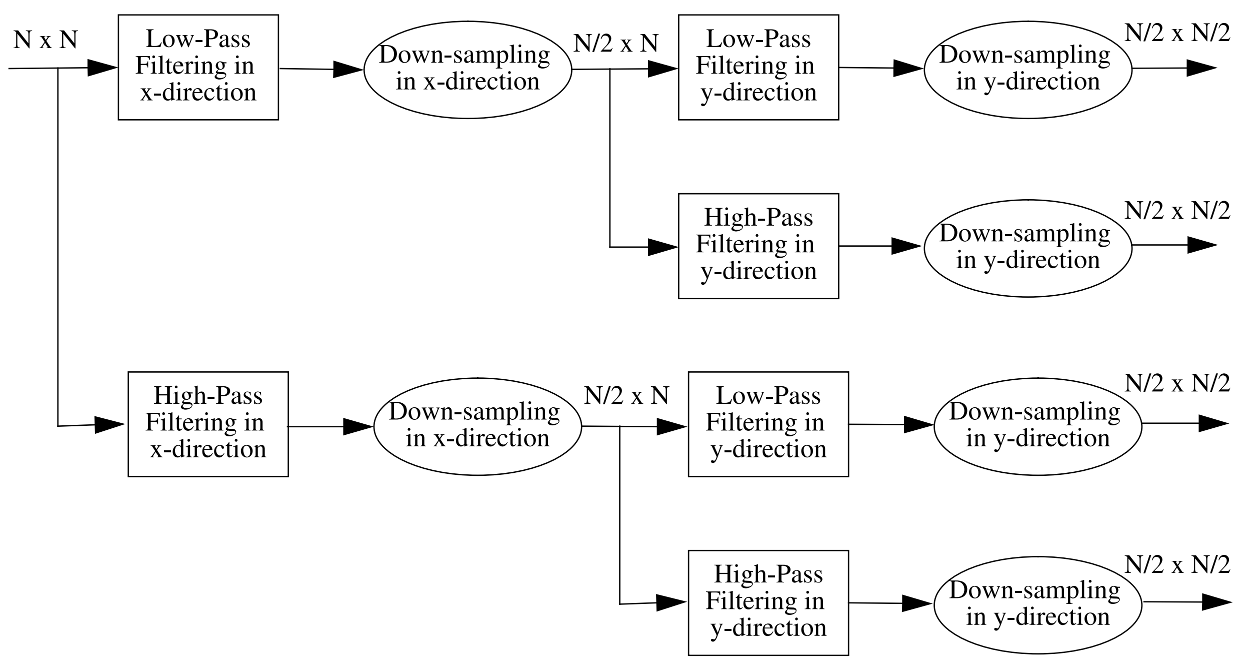

Figure 1. Single Stage of Spatial Subband Compression Scheme

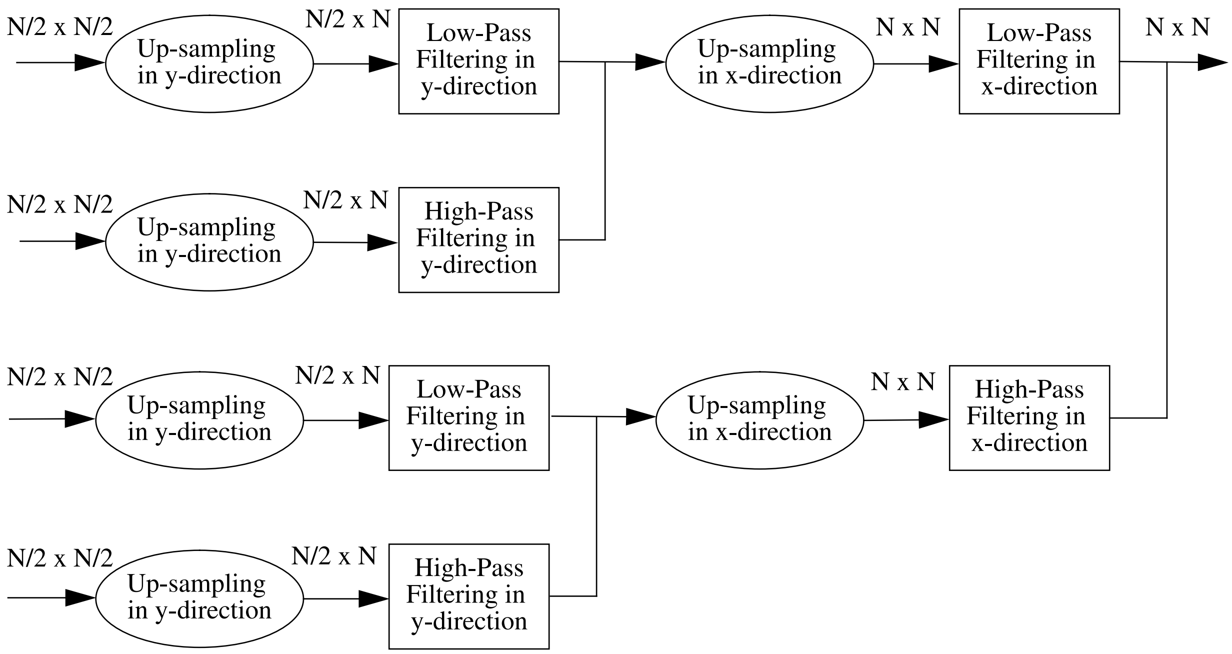

Figure 2. Single Stage of Spatial Subband Decompression Scheme

Additional compression can be achieved by cascading several stages of filtering and down-sampling. The implementation described in [Cha93b], for instance, cascades up to 10 stages, which produce nearly 40 output components. Any number of these output components can be combined to produce video streams of varying spatial and temporal resolution. This scheme produces a full-resolution compressed video stream with data rates comparable to other non-scalable algorithms such as MPEG [LeG91] (e.g., 3 Mb/s for subband coding vs. 2 Mb/s for MPEG). Thus, scalable compression inherently yields multiple representations of data (without the data replication needed by non-scalable strategies), while requiring bandwidth comparable to a single representation of the data.

Figure 3 shows an example scalable subband compression scheme that cascades three stages of spatial compression identical to the one shown in Figure 1. One of the four outputs of each stage is used as input to the next stage. Generally, the outputs of different stages could be combined to form hundreds or thousands of video streams with slightly different spatial resolutions. To simplify the presentation of disk layout policies, we restrict our study to four of the many possible output combinations. These form the successively lower resolution video streams shown in Figure 3: *Low* (L), *Medium* (M), *High* (H), and *Full* (F). The full-resolution video stream, which we assume has a throughput of 384 KB/s, is formed by combining all four outputs from the three stages. Successively lower resolution streams are formed by combining the four outputs of fewer stages. The lowest possible resolution, which has a throughput of 6 KB/s, is given by a single output component from the last stage.

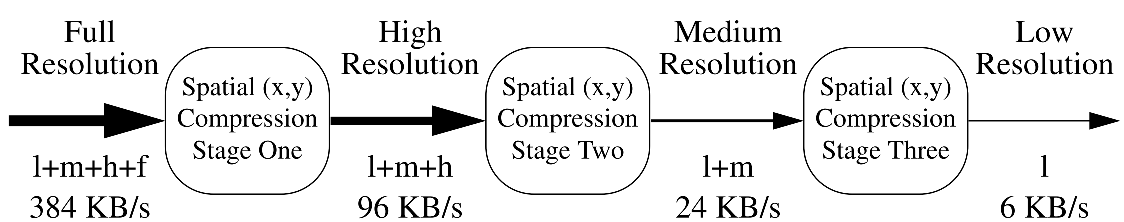

Figure 3. Subband Scalable Compression Example

Figure 4 shows a further simplification of the representation of the example subband scalable compression scheme. We say that the scheme has four *components* (l, m, h, f), which can be used to form the four supported video resolutions (Low (L), Medium (M), High (H), and Full (F)). The full-resolution video stream is produced by combining all four components. Successively lower quality streams are given by combining fewer and fewer of the components. The lowest possible resolution stream is given by the l component alone.

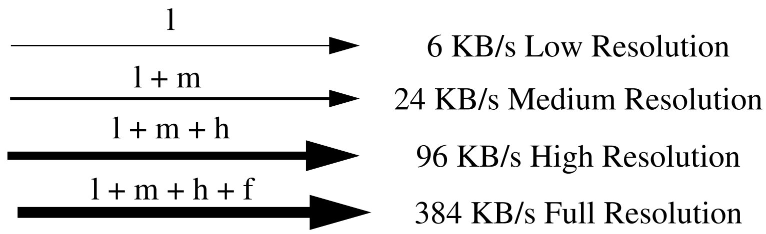

Figure 4. Simplified Scalable Compression Example

4.0 Multiple Resolution File System

We explore the use of a multiple resolution file system that uses knowledge of the video data structure as well as the storage system parameters to maintain multiple resolutions of each video sequence on-line. In this section, we focus on a more detailed discussion of video data structure.

4.1 Video Storage Structure

Continuous media data are structured by a logical playback unit (e.g., a single video frame), on which access patterns are based. To preserve this structure on disk, we define a *retrieval block* to be the smallest unit of video data that will be transferred from the disk array at one time. As in [Lou93, Ran91], a retrieval block corresponds to an integral multiple of logical playback units corresponding to a fixed playback duration for a given sequence. This playback duration is the same for all sequences in the system. We examine retrieval block sizes consisting of between one and sixty seconds' worth of data. The data for a given retrieval block is stored contiguously in the disk system, while the placement of successive retrieval blocks within a given stream is unrestricted.

This video storage structure is attractive for several reasons. First, the use of a fixed duration (rather than fixed size) retrieval block simplifies data transfer. Because many compression schemes perform variable bit rate compression, the size of video frames may vary. As a result, fixed size retrieval blocks may not only contain a different number of frames, but they may also contain partial frames. (To reconstruct these partial frames, two retrieval blocks must be transferred.) By allowing the size of the retrieval block to vary, fixed duration retrieval blocks ensure that an integral number of frames are stored contiguously. Furthermore, disk space allocation can be performed with increased fairness: video sequences with a high throughput are allocated larger retrieval blocks than streams with a low data throughput. (A drawback of a variable size, fixed duration retrieval block, however, is the potential for fragmentation of disk space.) Second, because each retrieval block corresponds to the same playback duration, scheduling of data retrieval can be simplified. This is discussed in further detail in the next section. Finally, because the storage of successive retrieval blocks is unrestricted, the policy allows for efficient interleaving of different video sequences.

4.2 Scheduling Data Retrieval

We assume that users make a single request for the retrieval of a given video sequence, and the actual retrieval of that data from disk is paced by the server. Once user requests have been accepted at the server, they are inserted into a schedule of video requests in the server scheduler. This scheduler uses a round robin policy in which data retrieval requests are performed in *rounds*, during which one retrieval block is read from the disk system for each of the user requests in the schedule. Scheduling of data retrieval is simplified because each retrieval block corresponds to the same fixed playback duration in the following way. Because enough data for the same playback duration is transferred for each user during each round, the relative ordering of deadlines remains the same from round to round. The retrieval schedule need not change, except when users are added or deleted. (This is in contrast with a fixed size retrieval block, where the transfer of one retrieval block per round corresponds to different playback durations for different users. To support continuous playback, the relative ordering of deadlines may change from round to round in this scheme.)

In order to maintain continuous playback at each client, the system must transfer the data for the next retrieval block before the current one has been completely displayed. As a result, the time between the start of successive scheduling rounds must correspond to the time for the client to play back a retrieval block. (If a scheduling round is shorter, data will be transferred faster than the client can display it, and data may accumulate at the client. If the rounds are longer, data for the next retrieval block may not be available at the client when the display of the current block has been completed.) In terms of scheduling rounds, the transfer of a retrieval block for a given user request must be completed by the following scheduling round. If the current request is still outstanding when the next retrieval block for that user is requested (in the following scheduling round), we say that the real-time playback deadline has been *missed*.

5.0 Disk Array Video Storage

The second component of our multiple resolution file system is knowledge of the storage system parameters. We propose the use of a disk array as the storage system, as other tertiary and secondary storage devices provide inadequate bandwidth, latency or capacity. Table 2 and Table 2 summarize several of the relevant metrics for alternative standalone storage technologies. The displayed metrics are original equipment manufacturer (OEM) price, capacity, transfer rate (for sustained transfers), seek time and rotational latency (for disk drives only). Table 3 illustrates several cost/performance metrics for alternative storage systems, including OEM price, capacity, number of drives, transfer rate, and access time (which corresponds to seek time for the disk arrays).

Technology	OEM Price	Capacity	Transfer Rate (MB/s)	Seek Time (ms)	Rotational Latency (ms)
Magnetic Disk Drives					
Toshiba MK-2428FB (2.5”)	$725	520 MB	6	12	7.5
Quantum Empire 540 (3.5”)	$695	540 MB	10	19	5.6
DEC StorageWorks (3.5”)	$3525	3.5 GB	5.2 to 6.9	12	5.6
Hitachi DK 517C-37 (5.25”)		3.6 GB	4.8	12.8	5.6
Optical Disk Drives					
Hewlett Packard C1716T	$4150	1.3 GB	1.6	23.5	12.5

Table 1. Cost/Performance Metrics for Standalone Disk Drives [McC94]

Technology	OEM Price	Capacity	Transfer Rate
Sony SDT-2000 DAT (4mm)	$1595	2 GB	183 KB/s
Exabyte EXB 8500 (8mm)	$2315	5 GB	470 KB/s
Metrum RSP-2150 (VHS)	$32,900	18 GB	2 MB/s

Table 2. Cost/Performance Metrics for Standalone Magnetic Tape Drives [McC94]

Technology	OEM Price	Capacity	No. of Drives	Transfer Rate	Access Time
Magnetic Disk Array					
Maximum Strategy S2P	$54,500	10.8 GB	10	18 MB/s	16 ms

Table 3. Cost/Performance Metrics for Alternative Storage System Technologies [McC94]

Technology	OEM Price	Capacity	No. of Drives	Transfer Rate	Access Time
Maximum Strategy Gen 4	$178,000	60 GB	up to 40	90 MB/s	11 ms
Storage Concepts Concept 151	$125,000	up to 324 GB	up to 216	up to 50 MB/s	14 ms
Optical Disk Libraries					
Hewlett Packard 200T	$96,000	187 GB	4	5.0 MB/s	3.7 s
Magnetic Tape Libraries					
Exabyte EXB10i	$13,000	50 GB	1	0.5 MB/s	20 s
Exabyte EXB120	$100,000	580 GB	4	2.0 MB/s	18 s
Metrum RSS-600	$266,000	10.8 TB	5	5.0 MB/s	8 s

Table 3. Cost/Performance Metrics for Alternative Storage System Technologies [McC94]

Tertiary storage devices, such as magnetic tape systems, provide adequate capacity (i.e., tens or hundreds of gigabytes, or terabytes) and, in some cases, adequate bandwidth (i.e., tens of megabytes per second) for the storage and retrieval of video data. They do not, however, provide sufficiently low latency or sufficiently high concurrency of access. Tapes generally incur a latency on the order of seconds or minutes during loading, followed by seconds to actually begin data retrieval. Tape systems provide poor concurrent access capabilities for multiple streams because there are typically only a few tape readers for many tapes. Switching between multiple tapes for a single reader requires that tapes be unloaded and reloaded, incurring tens of seconds' or minutes' worth of latency. For instance, Drapeau measures the cartridge switch time for the Exabyte EXB120 tape robot to be over four minutes, as shown in Table 4 [Dra93]. Even using a single tape with a single reader to satisfy multiple requests can be highly inefficient.

Operation	Time (sec)
Rewind time (1/2 tape)	75
Eject time	17
Robot unload	21
Robot load	22
Device load	65
Search (1/2 tape)	84
Total	284

Table 4. Components of Cartridge Switch Time for Exabyte EXB120 Robot [Dra93]

Repositioning tapes under the reader to satisfy an alternate stream's request often requires a complete rewinding of the tape and fast forwarding to reach the starting point for the alternate transfer. Thus, because of the high latency and low concurrency of access, tape systems are not a feasible choice for the sole storage medium of a large video storage system.

A single disk provides insufficient capacity (i.e., hundreds of megabytes, or gigabytes) and insufficient bandwidth (i.e., megabytes per second) for use in a large-scale video storage and retrieval system. In addition, a single disk cannot support many high-bandwidth concurrent users. We could use multiple disks, each with its own video sequence or movie, to support a large video system. This provides increased storage capacity and bandwidth, and allows an increased number of concurrent users. However, if some video sequences are more frequently accessed than others, their disks will become "hot spots" in the system. This load imbalance may prevent the system from utilizing its full bandwidth and concurrency capacities.

In order to provide the bandwidth, latency, and capacity needed for video storage and retrieval, we propose the use of a multi-disk system, or disk array, with each video sequence stored across multiple disks. Because different parts of different videos are stored on different disks, each disk contains pieces of several movies. This allows for better load balancing between frequently and infrequently accessed sequences.

5.1 Disk Array Architecture

We assume a two-dimensional disk array, as shown in Figure 5. The disks in each *column* communicate with the array controller using a shared bus accessed through a disk string protocol, such as SCSI. Data are distributed, or *striped*, across the disks that occupy the same position on each of the strings; this group is called a *row*. The *striping unit* describes the amount of logically contiguous data that is stored on a single disk [Che90]. Striping data over multiple disks distributes the requests for that data, preventing a single disk from becoming a bottleneck for frequently accessed data. Data are transferred in parallel across a row of disks. The disks in a single column perform independent, concurrent transfers. *Parallelism* describes the number of disks that service a single user request for data. The higher the degree of parallelism (i.e., the more disks used to service a request), the higher the transfer rate that the request sees. However, as more disks cooperate to satisfy a single request, fewer independent requests can be serviced simultaneously. The degree of *concurrency* in the system is defined as the average number of outstanding user requests being actively served at one time.

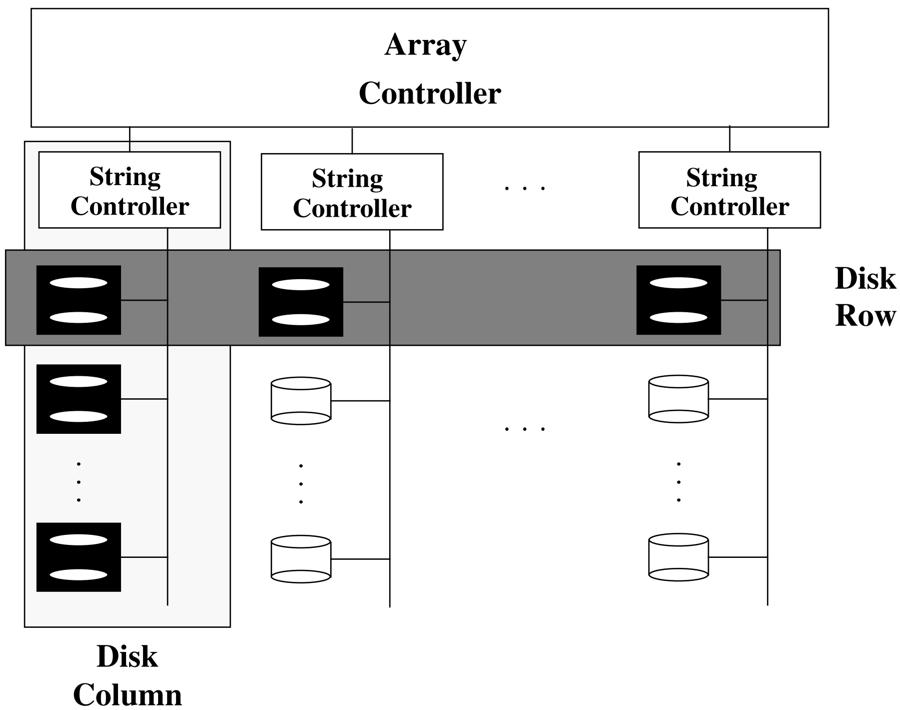

Figure 5. Two-dimensional Disk Array

Our disk array model, loosely based on the Berkeley RAID-II prototype [Lee92, Che93], assumes that each row contains eight disks, and that each column contains three disks, for a total of twenty-four disks. Each disk holds 320 MB worth of data, for a total of 7.5 GB of storage capacity.

5.2 Data Layout Strategies

We propose two strategies for the layout of multi-resolution video data. Each of these strategies varies the size of the striping unit to explore different degrees of the parallelism and concurrency offered by striping data across disks in a disk array. The first layout scheme, 1DISK, is shown in Figure 6. Each of the cylinders in the figure corresponds to one of the eight disks in a row of the array model. Each shaded block corresponds to a distinct retrieval block (from the same or different video streams). The letters within each shaded block (i.e., ‘l’, ‘m’, ‘h’, and ‘f’) show that the data comprising all resolutions of each retrieval block are stored contiguously. This scheme exhibits a low degree of parallelism, since each retrieval block is read from only a single disk. The scheme has a high degree of concurrency, because each column can simultaneously satisfy an independent request. Here, the striping unit size is equal to the size of a full-resolution retrieval block (e.g., 384 KB for a 1-second retrieval block).

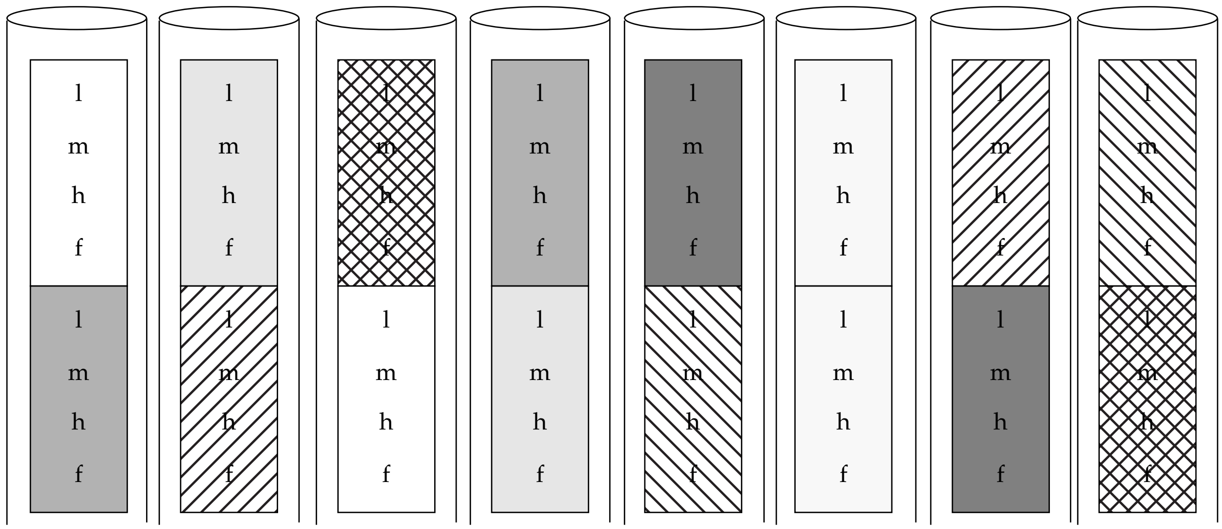

Figure 6. 1DISK Data Layout Scheme

Figure 7 shows the data layout for the second strategy, 8DISKS. Again, each different shading pattern corresponds to a distinct retrieval block. In this strategy, though, the data for each retrieval block are striped across the eight disks in a row. This scheme gives a much higher degree of parallelism for larger (e.g., high- and full-resolution) requests, because it transfers data from many disks for a given retrieval block. Due to this fact, the concurrency for large requests is limited. In this scheme, the placement of successive retrieval blocks' lower resolution components is rotated around the disks in a row. This allows the array to provide the same concurrency as the 1DISK scheme in satisfying lower (e.g., low- and medium-) resolution requests. Because data for a single retrieval block are striped across all of the disks in a row, the striping unit for this scheme is one-eighth of that for the 1DISK scheme (e.g., 48 KB for a 1-second retrieval block).

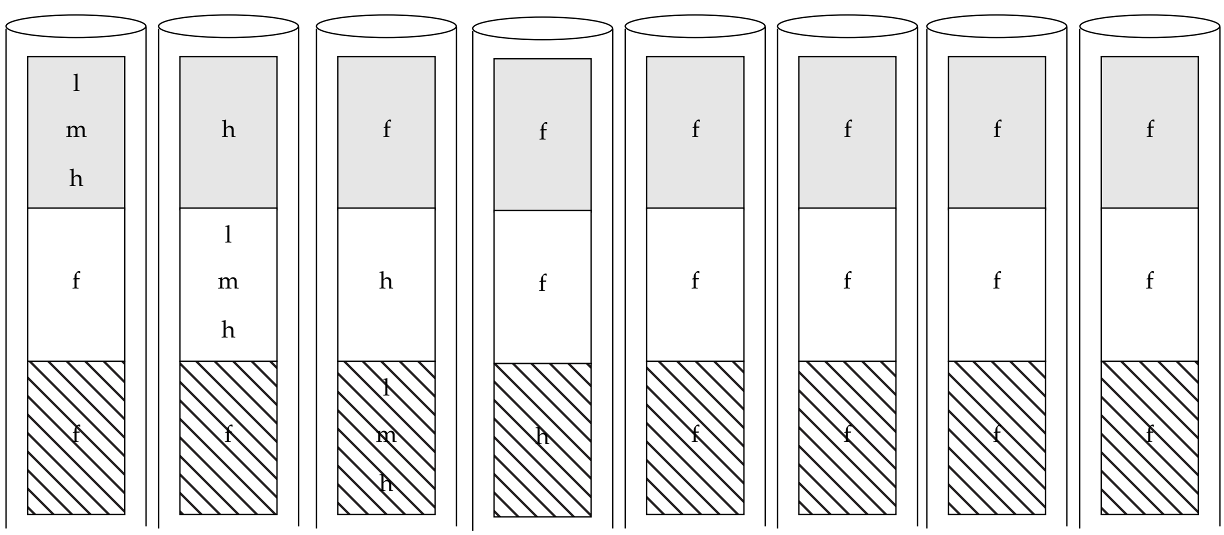

Figure 7. 8DISKS Data Layout Scheme

6.0 Simulation Study

To evaluate the performance of these data layout strategies and retrieval unit sizes, we have created an event-driven simulator of a RAID-style disk array using the Ptolemy simulation environment [Alm92]. This simulator is loosely based on the RAID-II storage server prototype developed at Berkeley [Lee92].

6.1 Ptolemy Simulation Environment

Ptolemy is a multi-domain environment produced at the University of California that can be used to simulate heterogeneous environments [Alm92]. It supports several different computational models, including data flow, discrete event, circuit simulation and code generation. A simulation in Ptolemy is decomposed into software modules called *blocks*, that are written in C++. These blocks are invoked at runtime in an order determined by the Ptolemy scheduler, and may exchange data among themselves (via message passing) as they execute. There are two block types: *stars*, which are elemental objects provided by user-written code, and *galaxies*, which are composed of stars and possibly other galaxies. The data exchanged between the blocks are divided into discrete units called *particles*. Hierarchies of stars and galaxies form a complete application called a *universe*.

For our simulation, we chose the discrete event domain, which provides support for event-driven simulation. In this domain, a particle represents an event that corresponds to a change in the system state. Each particle has an associated timestamp, which tells when it was generated in simulated time. All of the particles in the system are maintained in a global event queue, which is processed in chronological order by the scheduler. In particular, simulation occurs as the scheduler fetches a particle from the head of the event queue and sends it to the input of its destination block, where it is processed. After the block has executed, it may generate output particles, which are added to the global event queue. The scheduler continues fetching and dispatching events until a global stopping condition is met.

6.2 RAID-II Prototype

Our Ptolemy-based simulator is loosely based on the RAID-II storage server prototype, developed at Berkeley. The RAID-II prototype, shown in Figure 8, connects a SCSI-based disk array to a high-speed network in order to provide high-bandwidth file service [Lee92]. The central component of RAID-II is the “XBUS” crossbar controller, which connects the network, disks, and memory. The crossbar has four ports attached to VME busses; connected to each bus is an Interphase Cougar VME-SCSI disk controller. Each Cougar can drive up to two SCSI busses, or *strings*, each of which can support up to seven SCSI disks. A Sun 4/280 file server manages file metadata and control functions, using several point-to-point VME links to communicate with other system components. High-bandwidth transfers are accomplished by using the HIPPI source and destination network busses, while control and low-latency transfers are performed over an Ethernet interface. In its current configuration, RAID-II operates with four string controllers, each controlling two strings, for a total of eight strings. Each string supports three 320 MB IBM 0661 (“Lightning”) disks, yielding twenty-four disks, with 7.5 GB of total storage capacity.

6.3 Simulator Organization

The bulk of the simulator implementation was completed by the author in collaboration with Bruce Mah, another computer science graduate student, as a project for a graduate course in performance analysis. The Ptolemy simulator models several components of the video storage server architecture, including the disks, disk strings, string controllers, and file server host software. Because empirical studies performed with the RAID-II prototype [Che93] indicate that the disk subsystem is typically the bottleneck, we do not directly model the server memory and network. Lower-level components of the system, such as the disks, disk strings, and string controllers, are modelled closely, while higher-level server software is only approximately modelled.

The organization of the top-level simulator universe is shown in Figure 9. User requests for the playback of a given video sequence are accepted or denied based on the server admission control policy. Accepted user-level requests are passed on to the logical scheduler, which inserts them into the playback schedule. The logical scheduler periodically generates retrieval block requests for each of the user-level requests; these requests are directed to the physical scheduler. Because a single retrieval block may be composed of several disk blocks (e.g., for full-resolution data stored according to the 8DISKS layout scheme), the physical scheduler translates retrieval block requests into requests for actual disk blocks, which are sent to the string controllers for the target disks. (The organization of the disk string pair galaxy is shown in Figure 10.) The string controllers forward requests to the appropriate disk using the SCSI-based disk strings. Upon receipt of a request, a disk seeks and rotates to the appropriate disk position, and transfers the requested data. Once the transfer of the requested disk blocks has been completed, the disk sends a completion

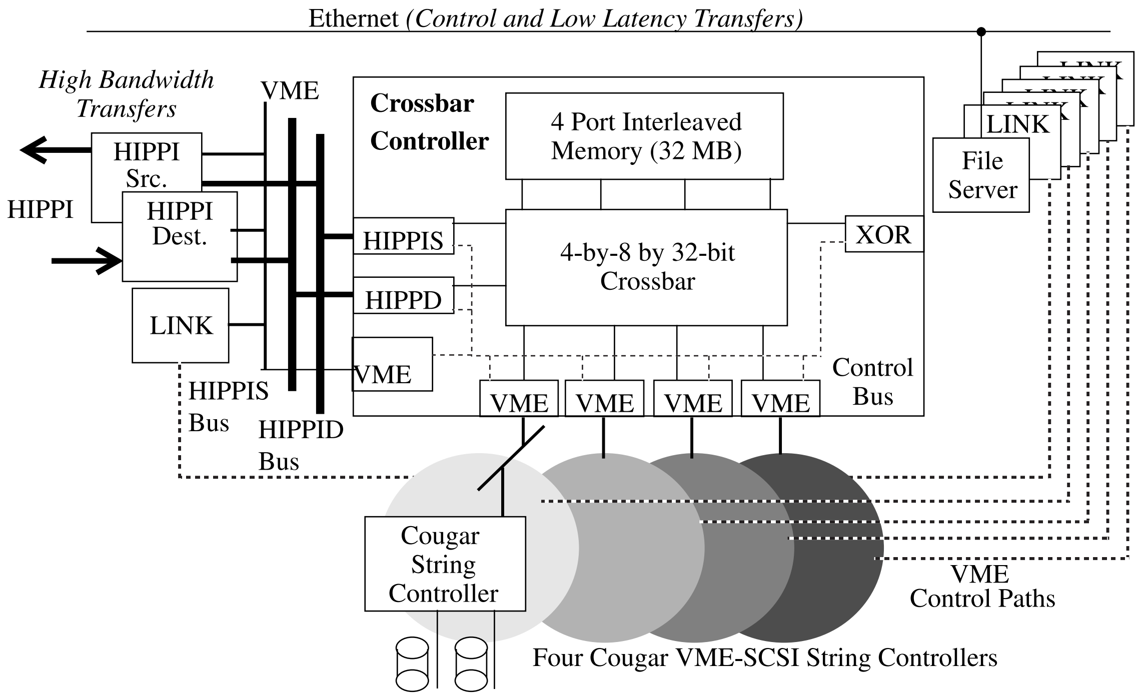

Figure 8. Organization of the RAID-II Storage Server

over the SCSI string to the string controller, which forwards the completion to the physical scheduler. When all of the disk blocks comprising one retrieval block have been successfully transferred, a completion is sent to the logical scheduler. The following sections describe each of the functional blocks, or stars, of the simulator in more detail.

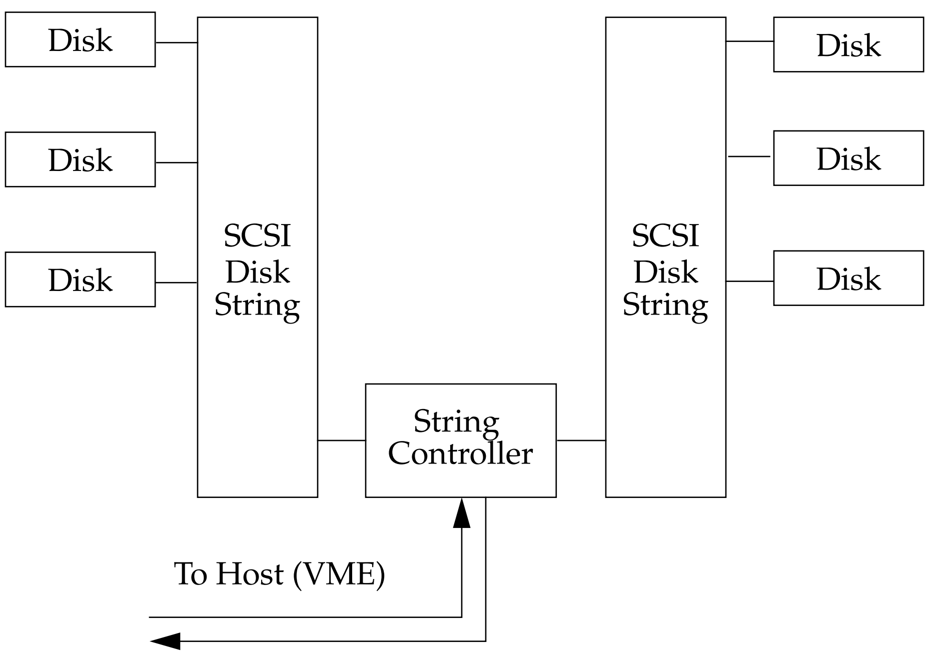

Figure 10. Organization of Disk String Pair Galaxy

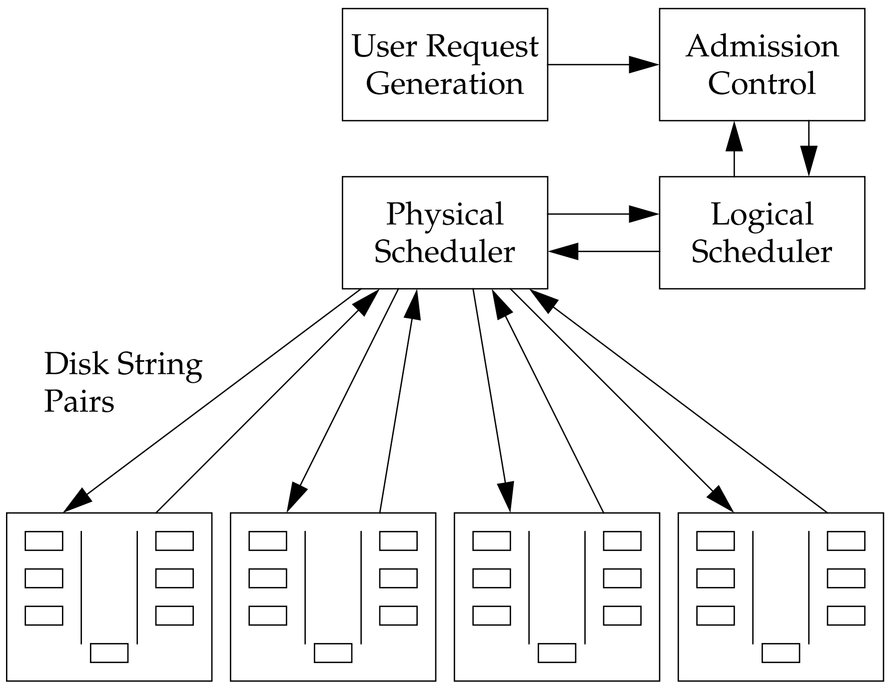

Figure 9. Organization of Top-level Simulator Universe

6.3.1 User Request Generation Star

User requests are characterized by an interarrival time, choice of video file, and choice of video resolution. User request interarrival was modeled using an exponential interarrival distribution plus a constant offset. (We include the constant offset to ensure that the user request interarrival time is sufficiently large.) Using three different Poisson arrival processes, we modeled five different load levels, which are shown in Table 5. We created file layouts for five video sequences, each having a ten minute duration. Each sequence was available in the four resolutions described in Figure 4: Low (6 KB/s), Medium (24 KB/s), High (96 KB/s), and Full (384 KB/s). In the modeled workload, users requested only full-resolution data, or chose uniformly between each of the four resolutions. Depending on the experiment, users chose between the video files according to a uniform distribution or according to a highly localized distribution called a Zipf's Law distribution [Knu73]. According to Zipf's Law, which has been used to accurately model library book borrowing patterns, the probability of choosing the nth most popular of V videos is $\frac{c}{n}$, where c is given by Equation 1.

Workload	**Exponential Mean (s)**	**Constant Offset (s)**
Very Infrequent	10	10
Infrequent	10	5
Moderate	10	1
Frequent	5	1
Very Frequent	1	1

Table 5. User Interarrival Workload Model

$$c = \frac{1}{1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{V}} \qquad \text{(EQ 1)}$$

The cumulative distribution given by Zipf's Law for $V = 24$ videos is shown in Figure 11.

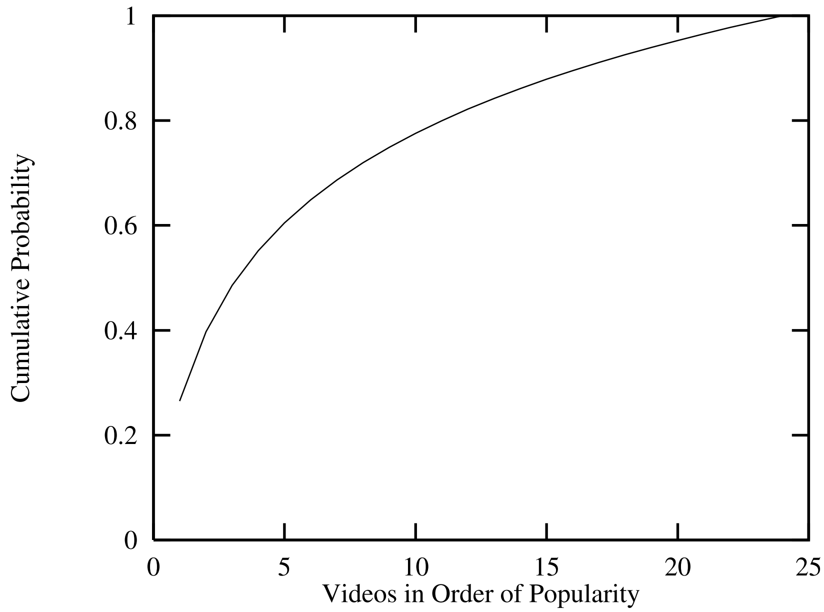

Figure 11. Zipf's Law Distribution for 24 Videos

To ensure that the simulation runs being compared received the same user inputs, the streams of user requests were captured and saved to several workload files. The user request generation star reads these files into the simulator at runtime, generating user requests, as shown in Table 6.

Message Type	Message Source	Message Indication	Action Taken	Output Message
User Request	User Workload File	New user request	Generate request for Admission Control	User-Request

Table 6. Message Particles Processed by User Request Generation Star

6.3.2 Admission Control Star

Generally, a video server system will accept or deny user requests, based on the server's ability to satisfy them. Several researchers have been successful in developing admission control policies for single-disk video server systems [Ran91, Lou93]. These schemes do not, however, easily translate into admission control schemes for complex disk array systems such as RAID-II. Because of the difficulty of deriving an effective disk array admission control scheme, we have focused our efforts on modelling the disk subsystem. Our simulator does not currently implement an admission control policy. Instead, each request is automatically "accepted" and forwarded on to the logical scheduler, as shown in Table 6.

6.3.3 Logical Scheduler Star

The logical scheduler is responsible for initiating the playback of video sequences by periodically requesting the transfer of video retrieval blocks from the disk system. Accepted user requests are inserted at the end of a schedule of requests, which is processed in round-robin order during a series of scheduling rounds. (Accepted requests are appended to the schedule to ensure that the server continues to meet the real-time deadlines of requests already in the

Message Type	Message Source	Message Indication	Action Taken	Output Message
User-Request	User Request Generation	New user request	Automatically accept request and notify logical scheduler of acceptance	Connection-Accept
Connection-Completion	Logical Scheduler	Video playback for user request has been completed	Release data structures associated with user request	n.a.

Table 7. Message Particles Processed by Admission Control Star

schedule. If the new request were inserted at the beginning or somewhere in the middle of the schedule, the user corresponding to a request processed after the new request might run out of data to play back in the time it takes the server to process the new request.) During a single round, the logical scheduler requests (from the physical scheduler) the transfer of one retrieval block for each of the accepted user requests. Several constraints must be satisfied to meet the user real-time playback requirements. First, the retrieval of a given block must be scheduled so that its data is available by the time that the previous block's playback has been completed. Second, to keep the disk system from saturation, the data for a given retrieval block must be completely transferred before the request for the next retrieval block is initiated. If this is not the case, we say that the deadline has been *missed*.

To address the first constraint, we limit the duration of a scheduling round to correspond to the duration of the retrieval block. For example, if a retrieval block corresponds to a second's worth of data, each scheduling round will be one second in duration. Our simulations use a total of five retrieval block durations: 1 second, 5 seconds, 15 seconds, 30 seconds, and 60 seconds. To address the second constraint, many systems use the admission control policy to limit the number of accepted users to prevent saturation of the system. As stated in Section 6.3.2, our system as modeled implements no admission control policy. As a result, under certain workload conditions, retrieval block requests miss their deadlines. These results are presented more fully in Section 7.0.

Table 6 describes the messages and events processed by the logical scheduler in managing the data retrieval for each of the user requests.

6.3.4 Physical Scheduler Star

The physical scheduler is responsible for translating logical scheduler requests for retrieval blocks to requests for ranges of actual disk blocks. In this sense, the physical scheduler is the module that implements the 1DISK and 8DISKS layout strategies described in Section 5.2. Upon receipt of a retrieval block request, the physical scheduler generates the corresponding disk block requests. For instance, one disk block request is generated for the 1DISK layout strategy, while eight disk block requests are generated for full-resolution data for the 8DISKS strategy. These requests are sent to the disk string controller that manages the destination disk(s). Once all of the disk blocks corresponding to the retrieval block have been transferred, the physical scheduler notifies the logical scheduler of the retrieval block completion. The messages used to manage the transfer of the disk block ranges for a given retrieval block are shown in Table 9.

6.3.5 Disk String Controller Star

The disk string controller model forwards disk block range requests from the physical scheduler to the appropriate disk via the disk string. The string controller maintains a request queue for each disk under its control (i.e., connected to one of the strings attached to the controller). These queues, which are processed in first-in-first-out (FIFO) order, permit the physical scheduler to have multiple disk block range requests outstanding for each disk, while enforcing a maximum of one outstanding request per disk.

Each disk queue can be in one of three states: *idle*, *contending*, or *active*. Upon arrival, disk block range requests are enqueued in the queue corresponding to the target disk. Regardless of the number of queued requests, a queue remains in the idle state until the controller chooses to contend on the queue's behalf for its string. (This choice is made based on the device number for the corresponding disk: the controller contends on behalf of the highest-numbered disk queue with an enqueued request. Device numbering and the choice of contenders are discussed in more

Message Type	Message Source	Message Indication	Action Taken	Output Message
Connection-Accept	Admission Control	New connection to be added to schedule	Add connection to end of retrieval schedule	n.a.
Retrieval-Block-Response	Physical Scheduler	Data for requested retrieval block has been transferred	Record retrieval block completion; if this was final retrieval block in sequence, release data structures associated with connection; inform admission control of user request completion	Connection-Completion
Timeout - beginning of scheduling round	Internal	Beginning of new scheduling round	Request retrieval block for first connection in schedule; initiate timeout to continue scheduling round	Disk-Block-Request; Timeout
Timeout - continuation of scheduling round	Internal	Continuation of current scheduling round	Request retrieval block for next connection in schedule; initiate timeout to continue scheduling round if there are additional connections	Disk-Block-Request; Timeout

Table 8. Message Particles and Events Processed by Logical Scheduler Star

Message Type	Message Source	Message Indication	Action Taken	Output Message
Retrieval-Block-Request	Logical Scheduler	Request for the retrieval of data corresponding to this retrieval block	Record request; request all disk block range(s) that correspond to the retrieval block	Disk-Block-Request(s)
Disk-Block-Response	String Controller	Data for requested range of disk blocks has been transferred	Record completion; inform logical scheduler if all disk block ranges for retrieval block request transferred; otherwise, wait for remainder of disk block responses	Retrieval-Block-Response

Table 9. Message Particles Processed by Physical Scheduler Star

detail in Section 6.3.6.) Once a disk queue is chosen to contend, the controller sends a String-Contend request to the disk string on behalf of the disk queue. When the contention request is granted, the queue's state becomes active; at this point the controller may transmit a request for the ranges of data blocks on the granted string. A request is completed once a Disk-Block-Response is received. This completion response can then be forwarded onto the physical scheduler. After all such request processing has been completed, the controller checks for idle strings, and contends for them on behalf of the highest-numbered queue with waiting requests. The messages that are processed by the disk string controller star are shown in Table 10.

Message Type	**Message Source**	**Message Indication**	**Action Taken**	**Output Message**
String-Grant	Disk String	Controller has been granted transmit privileges for string	Change disk queue state from contending to active; transmit disk block request	String-Read-Request
String-Read-Response	Disk String	Disk has completed its data transfer	Change disk queue state from active to idle; release data structures associated with request; forward disk block completion to physical scheduler; contend on behalf of next appropriate disk queue (changing its status to contending)	Disk-Block-Response
Disk-Block-Request	Physical Scheduler	Request for a range of disk blocks from a disk controlled by this controller	Enqueue request for the target disk	n.a.

Table 10. Message Particles Processed by Disk String Controller Star

6.3.6 SCSI-based Disk String Star

The SCSI disk string model permits communication between up to eight connected devices, such as disks or a disk string controller, numbered from zero to seven. (By default convention, the disk string controller is assigned device number seven. The three disks in our simulations are assigned device numbers zero, one, and two.) The SCSI string protocol is simulated by having each SCSI device contend for the shared bus for all transfers of control information and data. The SCSI string operates in one of two states: *idle* or *granted*. The bus is idle if there are no outstanding contention requests from any of the connected devices. A device contends for the bus by sending a StringContend request. The string chooses the winner of the contention by granting the bus to the highest priority (i.e., highest numbered) device. (This scheme reflects the unfairness inherent in the actual SCSI protocol: lower-numbered devices may never be granted the bus.) Once granted the bus, a device may proceed to transmit a message, which will be forwarded across the bus to its destination. The device may hold the bus only for the duration of its message transfer, after which time the bus is released back into the idle state. Thus, to perform subsequent transfers, the device must again contend for and be granted the bus. The message particles processed by the disk string star are shown in Table 11.

6.3.7 Disk Star

This disk model maintains a record of the disk head position, and simulates movement of the head to seek,

Message Type	Message Source	Message Indication	Action Taken	Output Message
String-Contend	Disk / String Controller	Device wishes to transmit on string	Grant access if currently idle; queue request if currently granted	String-Grant
String-Read-Request	String Controller	Controller requests range of disk blocks	Forward message if string already granted to controller; free string	String-Read-Request
String-Read-Response	Disk	Final message of disk's data transfer	Forward message if string already granted to controller; free string	String-Read-Response

Table 11. Message Particles Processed by SCSI-based Disk String Star

rotate and transfer data. This model is based on a model of an IBM Lightning disk drive obtained from Ed Lee and Pete Chen, former graduate student members of the RAID group [Lee89]. The parameters for this disk model are shown in Table 12. (Note: In this context, a sector corresponds to a disk block, as requested by the physical scheduler.)

Disk Parameter	Value
capacity	311 MB
bytes/sector	512
sectors/track	48
tracks/cylinder	14
cylinders/disk	949
average seek time (ms)	12.5
full rotation time (ms)	13.9

Table 12. Disk Model Parameters

We added to their model an approximation to track buffering. The reduction in rotational latency given by a track buffer is simulated in the following way. Rotational latency is only incurred if the head is positioned on a sector that was not requested. If the disk head is positioned anywhere in the requested range of disk blocks, no rotational delay is incurred. This approximate model for track buffering seems reasonable, because transfers will be done for retrieval blocks, which contain relatively large amounts of data, relative to the size of a track.

The capability to contend for and be granted the disk string was also added to the original disk model. After the disk receives a request for a range of disk blocks, it releases the bus to position the disk head. Once the head is positioned to begin the transfer, the disk contends for the string. Upon being granted access to the string, it begins to transfer data. To approximate the actual SCSI protocol, the disk is permitted to transfer only a track's worth of data. At this point, if additional data blocks must be transferred, the string must be released while the disk seeks to the next track. Once the head is positioned over the next track, the disk again contends for the string to continue the transfer. The messages used by the disk to receive requests and transfer disk block data are shown in Table 13.

7.0 Simulation Results

Four iterations of simulations were run for each combination of layout strategy, retrieval block duration, and user interarrival workload shown in Table 14. Statistics were collected on several quantities during the course of each simulation run; the average of the four simulation runs is presented for each of these metrics. We performed several experiments to evaluate the effectiveness of striping video data and storing multiple video resolutions, and to compare

Message Type	Message Source	Message Indication	Action Taken	Output Message
String-Read-Request	Disk String	Request to read range of disk blocks	Position head to start transfer of requested data	String-Contend when head positioned correctly
String-Grant	Disk String	Disk has been granted transmit privileges for string	Begin transfer of requested data	String-Read-Response

Table 13. Messages Processed by Disk Star

system performance for different retrieval block durations and different layout strategies. In Section 7.1, we compare striping video data versus storing each video on a single disk. Section 7.2 describes our measurements of the number of retrieval block missed deadlines for both single- and multi-resolution stored video data. To gain information on the utilization of the disks, we totalled the time spent by each disk in each of five phases: transferring data, seeking to the correct cylinder, rotating to the appropriate sector, contending for the SCSI bus, and idly awaiting a new request. We compare system performance for the different retrieval block durations, using service time and disk utilization as metrics in Section 7.3. Finally, in Section 7.4 we use service time to compare the performance of the two layout strategies. We focus our discussion on results for the 1DISK layout strategy, and present results for the 8DISKS scheme in an appendix, Section 12.0.

Simulation Input Parameter	Values
Layout Strategy	8DISKS, 1DISK
Retrieval Block Duration	1 sec, 5 sec, 15 sec, 30 sec, 60 sec
User Requested Resolution	full-resolution or uniformly between low, medium, high, and full
User Interarrival Workload	very infrequent, infrequent, moderate, frequent, very infrequent
User Requested Video	according to Zipf’s Law or uniform distributions

Table 14. Simulation Study Parameters

7.1 Effects of Striping Video Data

To understand the effects of striping video data across multiple disks in a disk array, we ran simulations with slightly different parameters than those described in Section 6.3.1. For this experiment, videos were roughly 14 minutes in length, corresponding to the length of movie compressed to 384 KB/s that can be stored entirely on a single 320 MB disk. User requests were generated using the infrequent interarrival workload, for full-resolution (384 KB/s) data using a 5-second retrieval block. Users chose among videos according to the highly localized Zipf’s Law distribution. Figure 12 shows the number of users that can be supported with no missed deadlines for both non-striped and striped data for several disk array sizes. In the non-striped case, each video is stored on a single disk, while for the striped case, video data is striped (using 5-second retrieval blocks) according to the 1DISK layout scheme. We see that striping data across the disks in the array permits service to considerably more viewers than storing each video on its own disk. This is because striping balances the load over all of the disks in the array, eliminating “hot spots” caused by numerous accesses to very popular video. For a 24-disk array, striped data allows nearly twice the viewers as non-striped data.

7.2 Effects of Multiple Resolution Video Data Storage

Figure 16 documents, for the 1DISK layout scheme, the relationship between the number of active requests in the system and the number of missed deadlines that the system generates, when multiple-resolution and single-resolution data are stored. Multiple resolutions corresponds to user requests which are uniformly distributed over the four different resolutions described in Section 3.2, while single resolution corresponds to only full-resolution user requests. The data shown are for 5-second retrieval blocks under moderate user interarrival load. Users chose uniformly

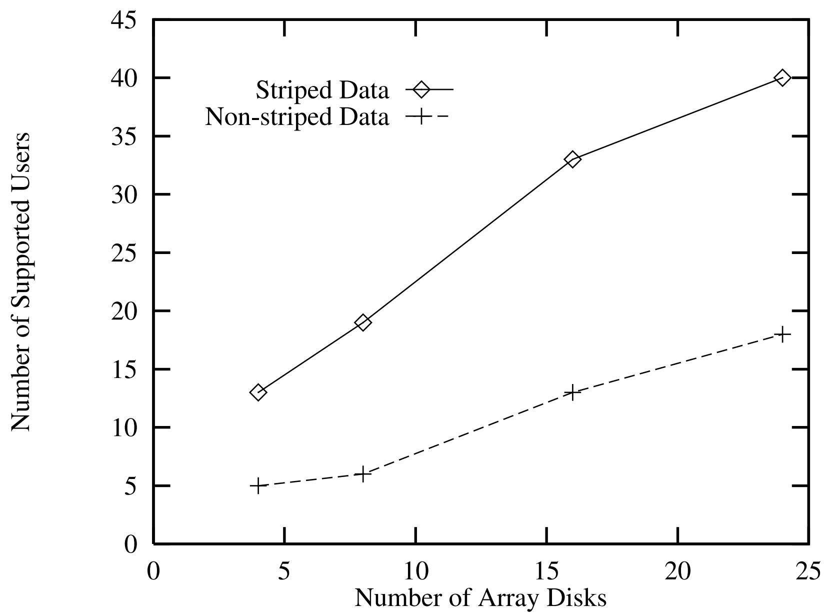

Figure 12. Number of Users Supported for Non-striped and Striped Data using the 1DISK Layout Scheme with 5-second Retrieval Blocks

between five 10-minute videos. The higher the number of active users when the first deadlines are missed, the more users the system can support while still maintaining real-time playback deadlines. The multiple-resolution system consistently meets all deadlines, while the single-resolution system misses successively more deadlines as the number of active requests increases.

For the single-resolution 1DISK layout scheme, presented in Figure 16, missed deadlines begin to occur at about 35 to 40 active users. (Analogously, for the 8DISKS scheme (shown in Figure 16), deadlines are missed beginning at about 50 active users. Other retrieval block durations exhibit similar behavior: the number of missed deadlines begins to increase starting at 35 to 55 users for the single-resolution case.) In all multiple-resolution cases, however, no deadlines are missed, even when the number of active users reaches a maximum of approximately 90 users. The occurrence of missed deadlines is due to the fact that each request is satisfied with full-resolution data, instead of data that matches the requested resolution. Thus, the multiple-resolution system can service a greater number of active users while still meeting all deadlines.

In general, it should be noted that the effect of missed deadlines is self-magnifying. As the retrieval time for a given retrieval request increases, the retrieval time for subsequent requests also increases, due to the added time these requests are queued. Thus, if a given retrieval request is delayed long enough to miss its deadline, there is a greater likelihood that subsequent requests will also miss their deadlines.

7.3 Effects of Retrieval Block Duration

Figure 17 shows the *overall service times* for the 1DISK case for all retrieval block sizes. Data are shown for uniform choice of video and uniform choice of resolution. We define overall service time as the time to transfer the 10-minute video sequence, measured as a sum of the times to request and receive each individual retrieval block. More specifically, this metric is the sum over all retrieval blocks of the delay between the issuance of a retrieval block request by the logical scheduler and the receipt of a completion message by the logical scheduler for that request. The lower the total average service time, the quicker that retrieval blocks can be requested and received, allowing more time to service other users during a given scheduling round. The figure shows that the overall service time generally decreases as the retrieval block size increases. The general decreasing trend is caused by the more efficient transfers that are possible for larger block sizes. A single larger block transfer eliminates the scheduling (e.g., scheduling, queueing, and disk positioning) overheads associated with several smaller block transfers. (An analogous decreasing total service time is observed for the 8DISKS case in Figure 17.)

Table 17 shows, for the 1DISKS strategy under a frequent user interarrival workload, the disk utilization as the percent of time, on average, that a disk spent in each of five phases: idle, transferring, seeking, contending, and rotat-

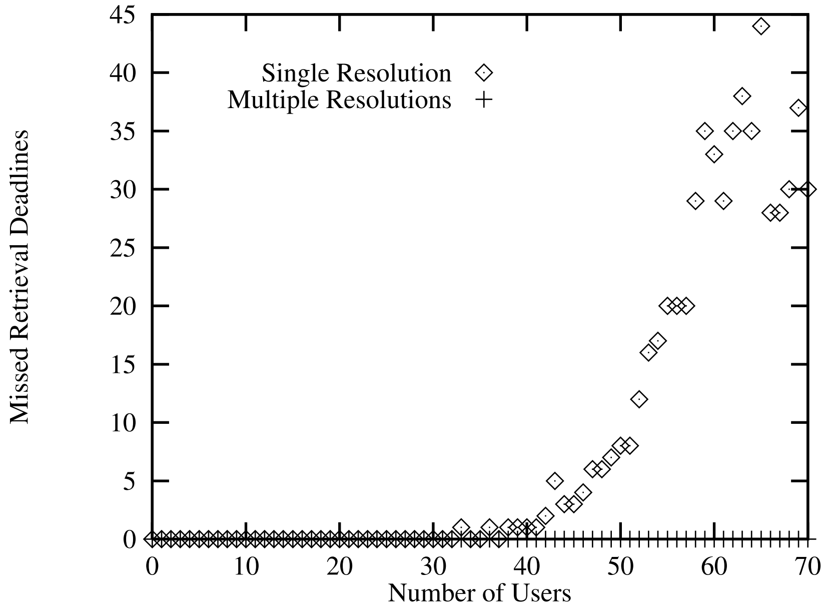

Figure 13. Missed Deadlines for Single- and Multiple-Resolution Systems using the 1DISK Layout Scheme with 5-second Retrieval Blocks

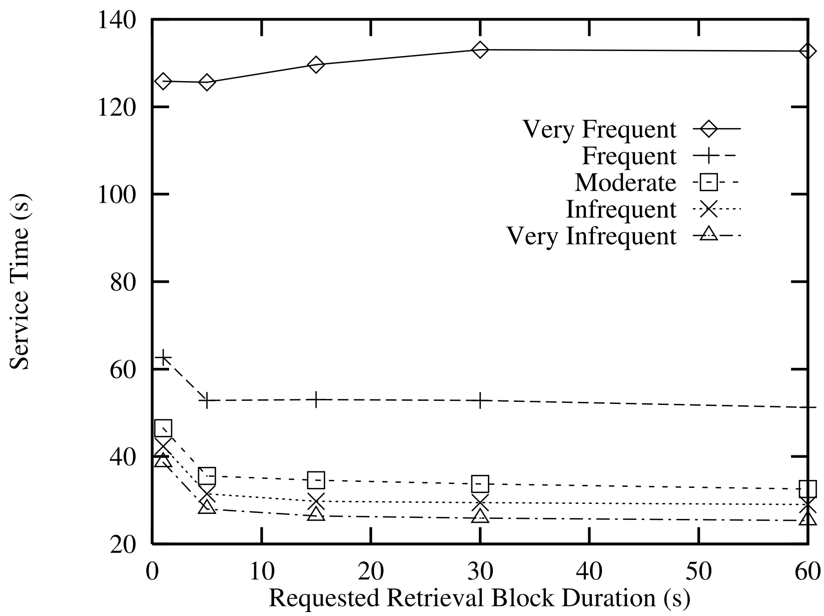

Figure 14. Total Service Time for 1DISK Layout Strategy

ing. (Similar disk utilization patterns have been observed for the 8DISKS scheme.) The amount of time spent positioning the disk heads (i.e., by seeking and rotating) decreases by 69 to 84 percent as the retrieval block duration increases from one to five seconds, and up to 60 seconds. As retrieval blocks grow longer, more data is stored contiguously, which corresponds to more efficient transfers. More specifically, there is less rotational latency to position the disk head over the data to be read when transferring longer retrieval blocks. In addition, a greater percentage of the seeks are between adjacent tracks, rather than between non-adjacent tracks, as needed for transferring several shorter (and non-contiguous) retrieval blocks. As the table indicates, most of the decrease in head positioning overhead is realized by the five-second retrieval block.

Disk Phase					
IDLE	81.8%	85.2%	85.8%	85.9%	85.7%
TRANSFER	9.1%	8.8%	8.8%	8.8%	8.8%
CONTEND	3.3%	4.2%	4.3%	4.4%	4.7%
SEEK	5.3%	1.7%	1.1%	0.9%	0.9%
ROTATE	0.5%	0.1%	0.0%	0.0%	0.0%
Retrieval Block Duration	**1 second**	**5 seconds**	**15 seconds**	**30 seconds**	**60 seconds**

Table 15. Percentage of Time Spent in Disk Phases for 1DISK Layout Scheme under Frequent User Interarrival Workload

7.4 Effects of Data Layout Policies

While moderately larger retrieval blocks allow for more efficient data transfers, they also require the presence of more buffer memory on the client, as shown in Table 16. Clients must buffer at least one retrieval block's worth of video data to ensure continuous playback. A reasonable compromise between efficient transfers and client buffering might be the 5-second retrieval block, which requires relatively little buffer space (e.g., 1.875 MB for five seconds' worth of 384 KB/s full-resolution data).

Retrieval Block Duration	Client Buffer Space
1 s	384 KB
5 s	1.875 MB
15 s	5.625 MB
30 s	11.25 MB
60 s	22.5 MB

Table 16. Required Client Buffer Space for a Single Stream of Full-Resolution Data

Figure 15 shows the relative total service time for the 8DISKS scheme with respect to the total service time for the 1DISKS scheme. (A ratio value of 1.0 means that the total service times for 8DISKS and 1DISK are equal.) In general, as the intensity of the user interarrival load increases, the 8DISKS scheme has worse relative performance. As more and more user requests enter the system, there is a greater likelihood that one of the eight disks needed to satisfy an 8DISKS retrieval block request will be temporarily unavailable, increasing the overall retrieval block latency. This is not the case for 1DISK retrieval block requests, since only a single disk must be accessed to complete the transfer. Thus, the 8DISKS scheme does not effectively support as high a degree of concurrency as the 1DISK scheme.

The 8DISKS scheme performs five to fifteen percent better for lighter user interarrival loads (i.e., very infrequent, infrequent, and moderate) and short to moderate-duration retrieval blocks (i.e., 5 to 30 seconds). For heavier interarrival loads (i.e., frequent and very frequent) and longer retrieval block durations (i.e., 30 and 60 seconds), however, the 1DISK scheme outperforms (or performs only slightly worse than) the 8DISKS scheme. Because of the added complexity of the 8DISKS scheme, and also because video servers will be expected to handle a large number of user requests, we anticipate that the 1DISK scheme will perform well.

7.5 Summary of Simulation Results

Overall, empirical evidence shows that striping video data over disks in an array can provide up to a two-fold increase in the number of viewers supported. In addition, the multiple resolutions provided by scalable compression allow a video file server to satisfy more user requests than a server that stores a single resolution of the video data. The use of moderately large retrieval blocks improves server performance, but requires more buffer space on the clients. Simulations show that the use of 5-second retrieval blocks is a reasonable compromise. Finally, the performance of the 1DISK scheme meets or exceeds that of the 8DISKS scheme for heavy workloads and long retrieval blocks.

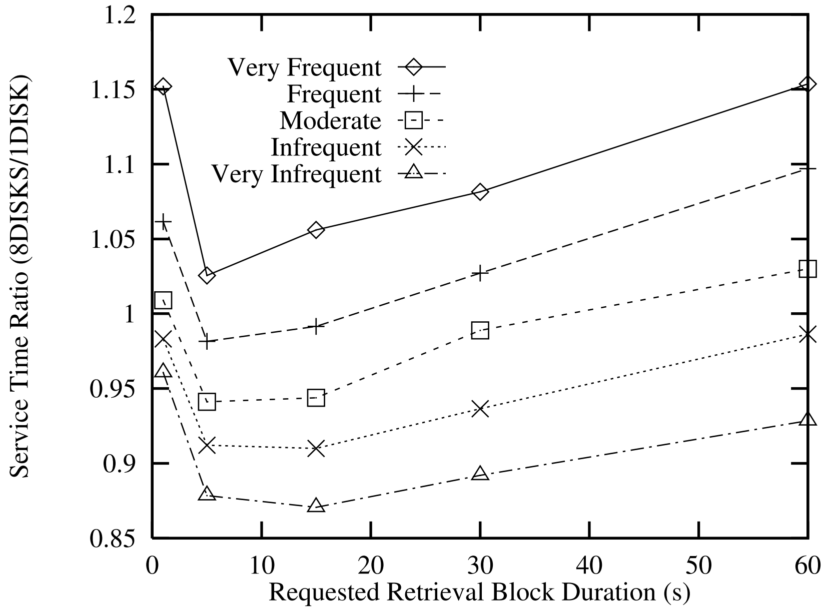

Figure 15. Ratio of Total Service Time for 8DISKS with respect to 1DISK

8.0 Summary and Remaining Work

Many modern file systems are unsuitable for the storage and retrieval of continuous media data such as video in heterogeneous client environments. To address these concerns, we propose an integrated video file system that uses knowledge of the disk system parameters as well as the video data structure to store multiple resolutions of each video sequence. This file system uses the high-bandwidth, high-capacity storage of a disk array to support many concurrent clients.

We have presented two layout strategies for multi-resolution data, which vary in the degrees of parallelism and concurrency they use to satisfy requests. Using event driven simulation, we have evaluated the performance of the proposed multiple resolution layout strategies. Our results show striping video data doubles the number of user requests that can be supported by a disk array-based video server. In addition, the storage of multiple resolutions allows the server to satisfy considerably more user requests than a system that stores only a single resolution data. The performance of the 1DISK scheme meets or exceeds that of the 8DISKS scheme for heavy workloads and long retrieval blocks.

A natural continuation of this work is to explore how the system would scale as additional components are added (e.g., more servers, more controllers per server, more string per controller, etc.). In addition, we would like to study how a magnetic disk arrays could be used as part of a hierarchical architecture, which includes tertiary storage, for providing video service. It would also be useful to derive a system admission control policy, similar to the ones presented in [Lou93,Ran91], for a disk array system. In our system, if new requests arrive during a period of system saturation, it may be possible to dynamically reduce the QoS provided to current requests (without violating client specifications) in order to free system resources that can then be used to satisfy the new client requests. Thus, the number of clients supported may be improved over that of other continuous media file services. Finally, it would be desirable to study the effects on disk block fragmentation brought about by the storage of data compressed by variable bit rate compression.

9.0 Availability

The source code for the Ptolemy-based disk array simulator described in this report is available via anonymous FTP from `ginger.Berkeley.EDU`. The relevant file is `/pub/kkeeton/videoSim94.tar.Z`. The Ptolemy source files are also included as an appendix to this report in The source distribution for the Ptolemy simulation package is available via anonymous FTP from `ptolemy.Berkeley.EDU` in directory `/pub`.

10.0 Acknowledgments

The bulk of the simulator implementation was completed in collaboration with Bruce Mah; the author wishes to thank him for his sizeable contributions. In addition, I wish to thank Ann Drapeau, Domenico Ferrari, Randy Katz, Bruce Mah, Dave Patterson, Srini Seshan and Ron Widyono for many insightful discussions during the development of these ideas. Support for this project was given by an AT&T Bell Laboratories Doctoral Fellowship, and by Department of Energy grant DE-FG03-92ER25135.

11.0 References

[Alm92] "The Almagest," Manual for Ptolemy, Version 0.4. University of California at Berkeley, 1992.

[Cha93a] E. Chang. Personal communication, October 1993.

[Cha93b] E. Chang and A. Zakhor. "Scalable Video Coding Using 3-D Subband Velocity Coding and Multirate Quantization," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, April 1993, pp. V:574 - 577.

[Che90] P. Chen and D. Patterson. "Maximizing Performance in a Striped Disk Array," *Proceedings of the 17th International Symposium on Computer Architecture*, May 1990.

[Che93] P. Chen, et al. "Performance and Design Evaluation of the Raid-II Storage Server," *International Parallel Processing Symposium Workshop on I/O in Parallel Computer Systems*, April 1993. Invited for submission to the Journal of Distributed and Parallel Databases.

[Che91] A. Chervenak Drapeau and R. H. Katz. "Performance of a Disk Array Prototype," *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1991, Vol. 19, pp. 188-197.

[Chi93] T. Chiueh and R. Katz. "Multi-Resolution Video Representation for Parallel Disk Arrays," *Proceedings of ACM Multimedi93*, June 1993, pp. 401 - 409.

[Dra93] A. Drapeau and R. Katz. "Striping in Large Tape Libraries," *Supercomputing* '93, November 1993.

[Kat91] R. Katz, T. Anderson, J. Ousterhout, and D. Patterson. "Robo-line Storage: Low Latency, High Capacity Storage Systems over Geographically Distributed Networks," Technical Report UCB/CSD 91/651, University of California at Berkeley, 1991.

[Knu73] D. Knuth. *The Art of Computer Programming*, Second Edition, Addison-Wesley Publishing Co., 1973.

[Lee89] E. Lee and P. Chen. Source code to `raidsim`, a RAID simulator, 1989.

[Lee92] E. Lee, et al. "RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service," Technical Report UCB/CSD 92/672, University of California at Berkeley, 1992.

[LeG91] D. Le Gall. "MPEG: a video compression standard for multimedia applications," *Communications of the ACM*, Vol. 34, No. 4, pp. 46-58.

[Lou93] P. Lougher and D. Shepherd. "The Design of a Storage Server for Continuous Media," *The Computer Journal,* Vol. 36, No. 1, 1993, pp. 32 - 42.

[McC94] M. McCarthy and B. Green. "Buyers' Guide Part 2: Mass-storage OEMs," *SunWorld*, 7:1, January 1994, pp. 69-90.

[McK84] M. McKusick, W. Joy, S. Leffler, and R. Fabry. "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, 2:3, August 1984, pp. 18 1- 197.

[Ran91] P. V. Rangan and H. Vin. "Designing File Systems for Digital Video and Audio," *Proceedings of the 13th Symposium on Operating Systems Principles*, *Operating Systems Review*, October 1991, pp. 81 - 94.

[Ros91] M. Rosenblum and J. Ousterhout. "The Design and Implementation of a Log-Structured File System," *Proceedings of the 13th Symposium on Operating Systems Principles*, October 1991, pp. 1 - 15.

[Woo91] J. Woods. *Subband Image Coding*, Kluwer Academic Publishers, 1991.

12.0 Appendix: Simulation Results for 8DISKS Layout Scheme

12.1 Effects of Multiple Resolution Video Data Storage

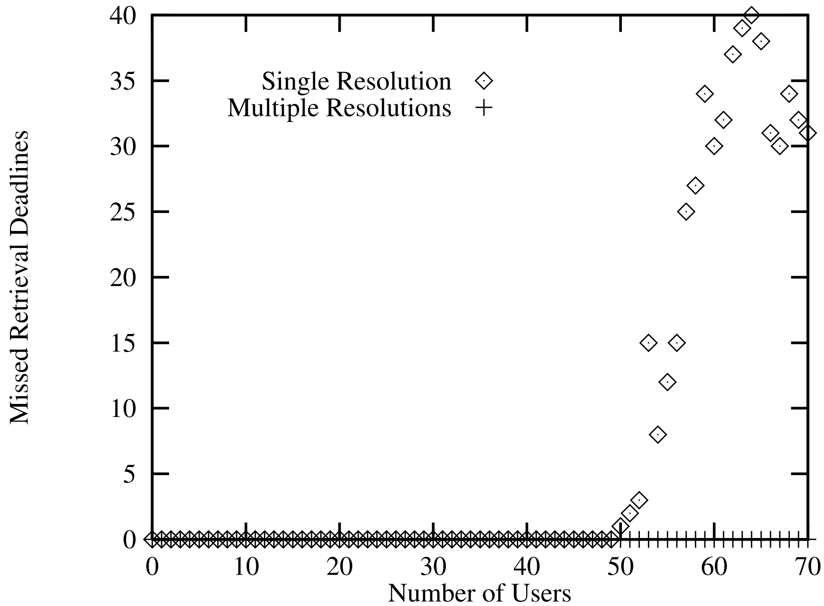

Figure 16. Missed Deadlines for Single- and Multiple-Resolution Systems using the 8DISKS Layout Scheme with 5-second Retrieval Blocks

12.2 Effects of Retrieval Block Duration

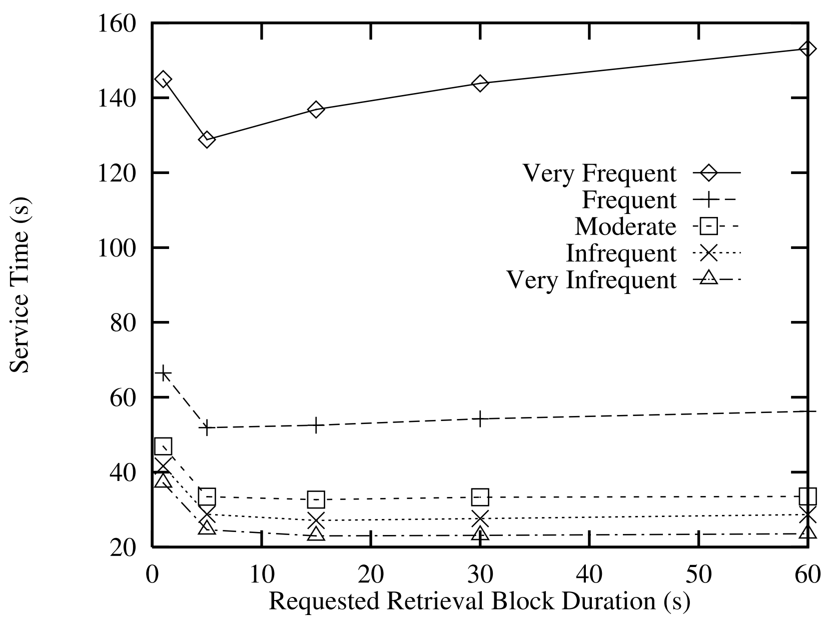

Figure 17. Total Service Time for 8DISKS Layout

13.0 Appendix: Ptolemy Source Code for Disk-Array Video Server

Disk Phase					
IDLE	80.3%	84.6%	85.4%	85.7%	85.6%
TRANSFER	9.1%	8.8%	8.8%	8.8%	8.8%
CONTEND	3.3%	4.6%	4.7%	4.6%	4.8%
SEEK	6.9%	2.0%	1.2%	1.0%	0.9%
ROTATE	0.5%	0.0%	0.0%	0.0%	0.0%
Retrieval Block Duration	**1 second**	**5 seconds**	**15 seconds**	**30 seconds**	**60 seconds**

Table 17. Percentage of Time Spent in Disk Phases for 8DISKS Layout Scheme under Frequent User Interarrival Workload