From the Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, England, UK, November 1993.

The Evaluation of Video Layout Strategies on a High-Bandwidth File Server

Kimberly Keeton, Randy H. Katz
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720
{kkeeton,randy}@cs.berkeley.edu

We propose a systems approach to providing video service which integrates the multi-resolution data generated by scalable compression algorithms with the high-bandwidth, high-capacity storage provided by disk arrays. The results of our simulations show that the storage of multi-resolution video permits service to considerably more users than the storage of single-resolution video. In addition, retrieval of data striped across the disks of an array can be performed much more efficiently than retrieval from a single disk.

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 which supports the playback of stored video to client display devices. Video storage servers must address 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 jitter 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 which 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 heterogeneity problem is to store multiple representations of video on the server. Each request can then be satisfied with a representation which 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 [6].

In this paper, we present an integrated systems approach to video service, which investigates the storage of multiple resolutions of video data on disk arrays. First, we discuss scalable compression algorithms, which generate video streams with multiple resolutions and rates. We also introduce several data layout strategies for disk arrays, which use information about the video data structure and disk array parameters to store video data for efficient playback. Finally, we present results from a simulation to evaluate the relative performance of these data layout strategies.

2.0 Related Work

Video playback generally requires periodically scheduled, long sequential reads under strict delay requirement. Existing file system data layout strategies support this access pattern to varying degrees. Traditional file systems, such as the BSD Fast File System [8] and Sprite LFS [10] use layout strategies which 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 [9]. 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 [7]. 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 [5].

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 vary widely. For instance, workstations utilize large displays capable of holding multiple resizable windows, while portable computers use much smaller physical displays, capable of supporting smaller windows. Because the server has a single version, full-resolution data is 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 work to dynamically adjust the quality of service parameters (in addition to decompressing the compressed stream).

Scalable compression algorithms address these issues. This class of compression algorithms produces one compressed full-resolution video bit stream. It uses a hierarchical format where various subsets correspond to different resolutions or rates of the same video sequence. Servers which 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 subsets 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 compression uses subband coding [11]. The implementation described in [2] produces a compressed bit stream which requires only slightly more storage (e.g. 1.5 times) than other non-scalable algorithms such as MPEG. Thus, scalable compression inherently yields multiple representations of data (without the data replication needed by non-scalable strategies), while requiring storage comparable to a single representation of the data.

Figure 1 shows an example scalable compression scheme. This scheme has four components (l, m, h, f), which can be used to form four *subsets*: *Low* (L), *Medium* (M), *High* (H), and *Full* (F). The full-resolution video stream, which has a throughput of 384 KB/s, 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, with a throughput of 6 KB/s, is given by the l component alone.

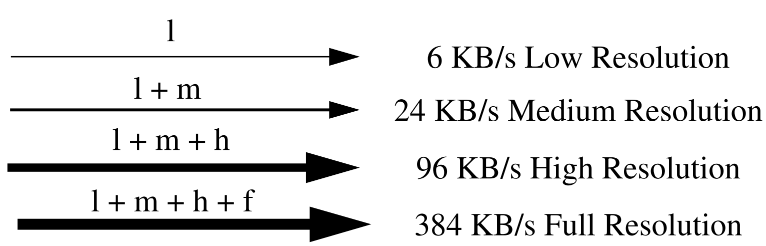

Figure 1. Scalable Compression Example

4.0 Multiple Resolution File System

We explore the use of a multiple resolution 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. Multiple resolutions will allow the system to more closely address the different qualities of service requested by a heterogeneous client population.

4.1 Video Data Structure

Continuous media data are structured by a logical playback unit (e.g. 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 [7, 9], a retrieval block corresponds to an integral multiple of a stream's logical playback unit corresponding to a fixed playback duration. We examine retrieval block sizes between one and sixty seconds' worth of data. The data for a given retrieval block is stored contiguously in the disk system. The placement of successive retrieval blocks within a given stream, however, is unrestricted. This policy allows for efficient interleaving of multiple video sequences.

4.2 Scheduling Data Retrieval

We assume that users request the retrieval of a given duration of a video sequence for a given duration, and the actual retrieval of the 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 a single *scheduling round,* one retrieval block is read from the disk system for each of the user requests in the schedule. In order to meet the real-time requirements of the video streams, the transfer of a retrieval block for a given user request must be completed by the next scheduling round. If the retrieval is still outstanding when the user request is next scheduled, the real-time deadlines will be missed.

5.0 Data Layout on Disk Arrays

We propose several strategies for the layout of multi-representation video data. Each of these strategies explores different aspects of the *parallelism* and *concurrency* offered by striping data across disks in a disk array. 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.

5.1 Disk Array Architecture

We assume a two-dimensional disk array, as shown in Figure 2. The disks in each *column* communicate with the array controller using a shared bus accessed through a disk string protocol, such as SCSI. Data is distributed, or *striped*, across the disks which 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 [3]. Data is transferred in parallel across a row of disks. The disks in a single column perform independent, concurrent transfers. Our disk array model, loosely based on the Berkeley RAID-II prototype [4, 6], assumes that each row contains eight disks, and that each column contains three disks, for a total of twenty-four disks.

5.2 Data Layout Strategies

We propose several data layout strategies which vary the size of the striping unit to provide different degrees of parallelism and concurrency. The first layout scheme, 1DISK, is shown in Figure 3. Here, each of the cylinders 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). 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. 386 KB for a 1-second retrieval block).

Figure 4 shows the data layout for the second strategy, 8DISKS. Here, each different shading pattern corresponds to a distinct retrieval block. 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

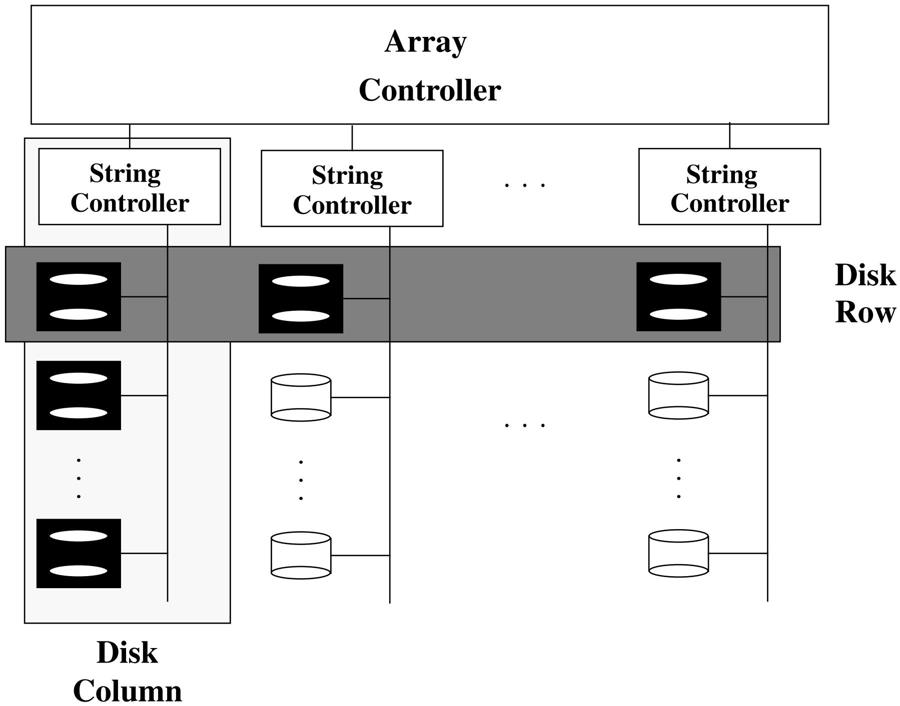

Figure 2. Two-dimensional Disk Array

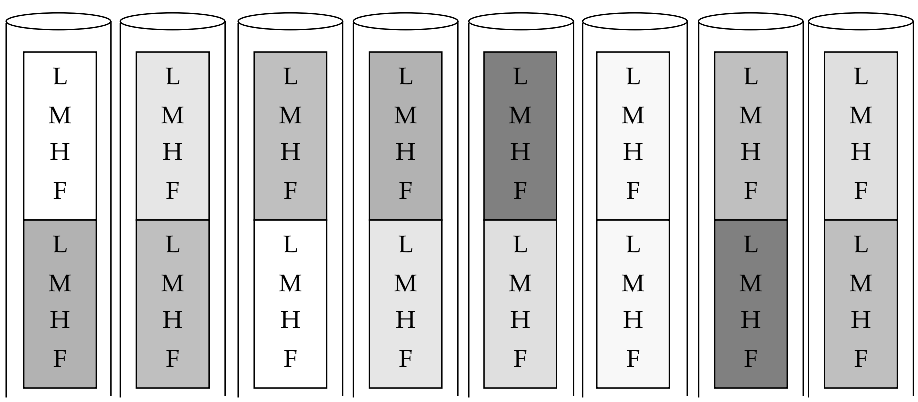

Figure 3. 1DISK Scheme

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 is 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).

6.0 Simulation Study

To evaluate the performance of these data layout strategies, we have created an event-driven simulator of a RAID-style disk array using the Ptolemy simulation environment [1]. It models the video server architecture, including the disks, disk strings, string controllers, and file server host software. Because the disk subsystem is typically the bottleneck in a disk array I/O system [4], we do not directly model the server memory and network. The disk model calculates seek time, rotational latency, bus contention time and transfer time for disk operations, using an approximation

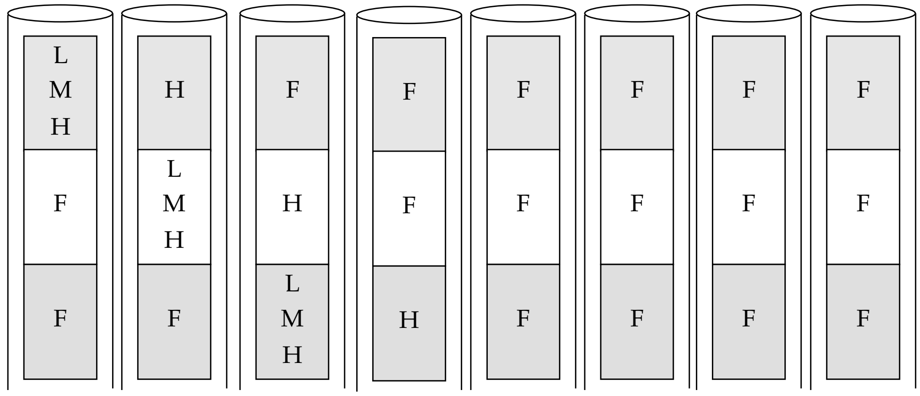

Figure 4. 8DISKS Scheme

to track buffering. The parameters for this disk model are shown in Table 1. The SCSI string protocol is simulated by having each disk contend for the shared bus for all transfers. The string controller model implements command queues for the disks, which allows the host to have multiple SCSI requests outstanding for each device. Higher-level components, such as the host scheduler software, are modeled as queueing systems with approximations for the service times.

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 1. Disk Model Parameters

We created file layouts using the 1DISK and 8DISKS strategies for five video files. Each sequence had a ten-minute duration, and was available in the four resolutions described in Figure 1: Low (6 KB/s), Medium (24 KB/s), High (96 KB/s), and Full (384 KB/s). Layouts were generated for five retrieval block sizes: 1 second, 5 seconds, 15 seconds, 30 seconds, and 60 seconds.

User behavior was modeled using an exponential interarrival distribution plus a constant offset. Using three different Poisson arrival processes, we modeled five different load levels, which are shown in Table 2. Users chose uniformly between each of the five video files, and uniformly between each of the resolutions.

Workload	Mean Interarrival (s)	Constant Offset (s)
Very Low	10	10
Low	10	5
Medium	10	1
High	5	1
Very High	1	1

Table 2. User Workload Model

7.0 Simulation Results

Statistics were collected on several quantities during the course of each simulation run. In order to measure the amount of latency experienced by data, we measured the service time, the delay between the issuance of a retrieval request by the scheduler and the receipt of a completion message from the disk system for that request. We also counted the number of retrieval blocks which missed their deadlines. 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 also measured queue lengths for each disk, as well as for the scheduler. Due to space limitations, we focus our discussion of the results on the service time and missed deadline metrics.

Figure 5 documents 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. The data shown is for the 8DISKS scheme for a 5-second retrieval unit under medium load. (Similar behavior is exhibited for other workloads and retrieval unit sizes.) The resolutions of user requests are uniformly distributed. The multiple-resolution system consistently meets all deadlines, while the single-resolution system misses successively more deadlines as the number of active requests increases. This is due to the fact that each request is satisfied with full-resolution data, instead of data which matches the requested resolution. Thus, the multiple-resolution system can service a greater number of active users while still meeting all deadlines.

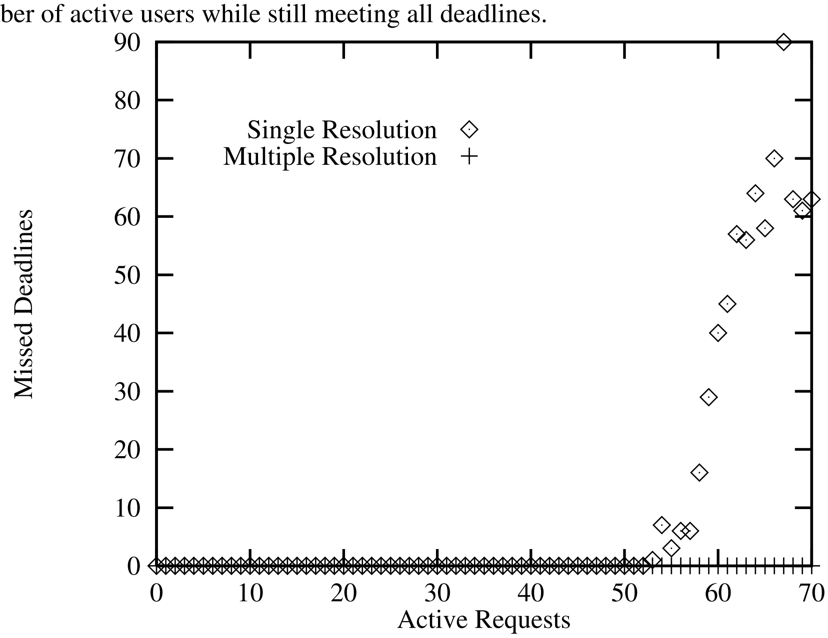

Figure 5. Missed Deadlines for Single- and Multiple- Resolution Systems

Figure 6 shows the overall service times for the 8DISKS case for all retrieval block sizes. This metric is the overall time to transfer the 10-minute clip, measured as a sum of the times to request and receive each individual retrieval block. The overall service time decreases as the retrieval block size increases to 15 seconds. There is an increase in the service time for the 30-second retrieval block, followed by a decrease for the 60-second block size. 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. However, the striped data for larger (e.g. 15-, 30- and 60-second) retrieval blocks spans multiple disk cylinders, requiring several seeks to retrieve a single block. (The striping units for these cases are shown in Table 3.) For the 15-second case under very high load, the efficiency of a larger transfer dominates the effect of the seeks. (For lower and moderate loads, these effects nearly cancel one another.) As the striping unit size increases for the 30-and 60-second cases, the cost of the seeks begins to dominate the transfer efficiency.

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 4. 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 full-resolution data).

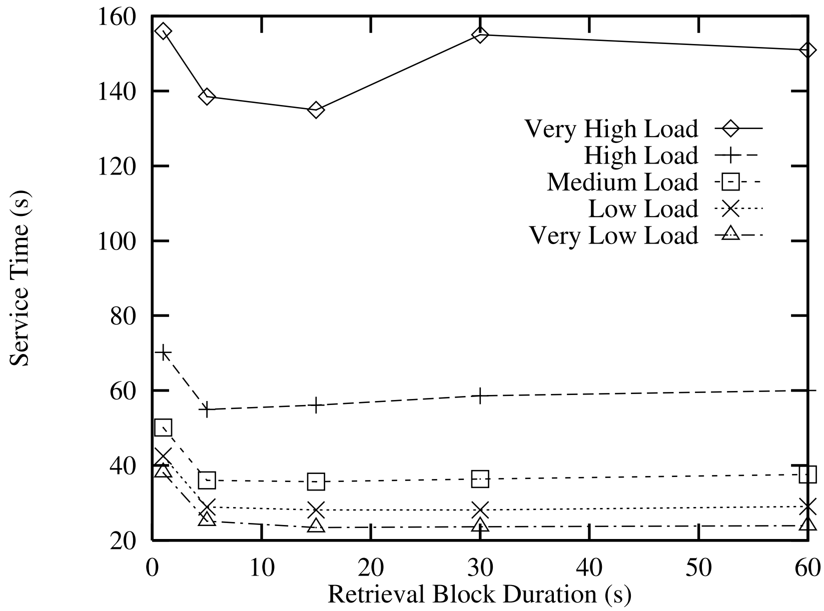

Figure 6. Total Service Time for 8DISKS Layout Policy

Retrieval Block Size (s)	8DISKS Striping Unit
1	48 KB
5	240 KB
15	720 KB
30	1440 KB
60	2880 KB

Table 3. 8DISKS Data Striping Unit Sizes

Retrieval Block Size (s)	Client Buffer Space
1	384 KB
5	1.875 MB
15	5760 KB
30	11.25 MB
60	22.5 MB

Table 4. Required Client Buffer Space for Full-Resolution Data

We see the effects of decreasing service times on missed deadlines in Table 5 and Table 6. These tables present performance results for full-resolution requests under low system load for the 8DISKS and 1DISK schemes, respectively. As the average overall service time decreases, the amount of time each request is queued waiting to be serviced declines. (An artifact of this reduced time spent in the system is that the average number of user requests decreases slightly.) This reduces the likelihood that a missed deadline will occur, and hence reduces the number of missed deadlines, as shown in the tables.

Table 5 and Table 6 also allow us to compare the performance of the 8DISKS and 1DISK layout schemes for full-resolution requests. For all retrieval block sizes, the average overall service time for the 8DISKS cases is an average of 30% less than that for the 1DISK case. This lower service time is due to the higher parallelism given by striping data across multiple disks in the 8DISKS scheme. As a result, the 8DISKS scheme experiences considerably fewer missed deadlines than the 1DISK scheme.

In general, it should be noted that the effect of missed deadlines is self-magnifying. As the retrieval time for a

Retrieval Block Size (s)	Average # Requests	Max # Requests	Average Service Time (s)	Missed Deadlines
1	37.9	49	219.5	822
5	37.8	49	183.9	5
15	37.5	49	184.5	6
30	37.3	49	181.7	0
60	36.5	49	174.8	0

Table 5. Performance Parameters for 8DISKS Scheme for Full-Resolution Requests at Low User Load

Retrieval Block Size (s)	Average # Requests	Max # Requests	Average Service Time (s)	Missed Deadlines
1	37.9	49	311.8	20247
5	37.8	49	303.7	2621
15	37.8	49	300.9	827
30	38.4	51	318.5	535
60	37.4	50	275.6	111

Table 6. Performance Parameters for 1DISK Scheme for Full-Resolution Requests at Low User Load

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.

Although not quantitatively shown here, the performance (with respect to service times and missed deadlines) of the 8DISKS and 1DISK schemes for low- and medium-resolution requests is identical, due to their identical degrees of parallelism and concurrency. The service time and missed deadline parameters for the two strategies begin to diverge with the high-resolution requests, due to the increased parallelism given by data striping.

Overall, empirical evidence shows that the multiple resolutions provided by scalable compression allow a video file server to satisfy more user requests than a server which stores a single resolution of the video data. In addition, 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 are a reasonable compromise. Finally, the performance of the 8DISKS scheme meets or exceeds that of the 1DISK scheme for all resolutions and retrieval block sizes.

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 which 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 used to satisfy requests. Using event driven simulation, we have evaluated the performance of the proposed multiple resolution layout strategies. Our results show that the storage of multiple resolutions allows a video server to satisfy considerably more user requests than a system which stores only a single resolution data. In addition, the 8DISKS layout scheme outperforms the 1DISK scheme on the basis of data latency and ability to meet real-time deadlines.

In the future, we will continue to evaluate the relative performance of the various data layout strategies. More specifically, we will explore the derivation of a system admission control policy similar to the ones presented in [7, 9]. 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.

Acknowledgments

The authors wish to thank Domenico Ferrari, Bruce Mah, Dave Patterson, Srini Seshan and Ron Widyono for their help during the development of these ideas. Kimberly Keeton was supported by AT&T Bell Laboratories Doctoral Fellowship, and by Department of Energy grant DE-FG03-92ER25135.

9.0 References

1. "The Almagest," Manual for Ptolemy, Version 0.4. University of California at Berkeley, 1992.
2. 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.
3. P. Chen and D. Patterson. "Maximizing Performance in a Striped Disk Array," *Proceedings of the 17th International Symposium on Computer Architecture*, May 1990.
4. 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.
5. T. Chiueh and R. Katz. "Multi-Resolution Video Representation for Parallel Disk Arrays," *Proceedings of ACM Multimedia '93*, June 1993, pp. 401 - 409.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. J. Woods. *Subband Image Coding*, Kluwer Academic Publishers, 1991.