The SPIFFI Scalable Video-on-Demand System*

Craig S. Freedman David J. DeWitt
Computer Sciences Department
University of Wisconsin
Madison, WI 53706
{freedman,dewitt}@cs.wisc.edu

Abstract

This paper presents a simulation study of a video-on-demand system. We present video server algorithms for real-time disk scheduling, prefetching, and buffer pool management. The performance of these algorithms is compared against the performance of simpler algorithms such as elevator and round-robin disk scheduling and global LRU buffer pool management. Finally, we show that the SPIFFI video-on-demand system scales nearly linearly as the number of disks, videos, and terminals is increased.

1 Introduction

In recent years, real-time digital multimedia systems have moved from concept to reality due to major advances in disk, network, and data compression technologies. In particular, video-on-demand systems capable of delivering any of a large selection of movies to a subscriber's television at the press of a button have attracted a great deal of attention. The operating constraints on a video-on-demand system are enormous. Multiple uninterrupted streams of video data must be delivered to a large number of clients. If the system fails to meet its real-time deadlines, one or more clients will experience a service interruption or "glitch."

The large size, rigid real-time constraints, and general complexity of a video-on-demand system leads to many difficult design decisions that may substantially affect a system's ultimate cost and performance. If poor decisions are made during a system's design, subscribers may experience frequent glitches or long waits to view movie selections. Alternatively, a poorly designed system may wind up costing substantially more than a well designed system while servicing the same number of customers. Given the alternatives already available to consumers (e.g., a trip to the local video store or movie theater), a video-on-demand system must be well designed and economically competitive to succeed in the marketplace.

This paper presents a simulation study of a video-on-demand system to determine the effect of varying a wide variety of parameters and design alternatives. We demonstrate the benefits of declustering or striping videos across many disks; we compare the performance of real-time disk scheduling against simpler algorithms such as elevator or round-robin; and we present buffer pool page replacement and prefetching algorithms that minimize memory requirements.

The remainder of this paper is organized as follows. Section 2 provides an overview of the minimum features and performance that we believe any video-on-demand system must provide. Section 3 reviews some related work while Section 4 describes our approach to the video-on-demand problem. Section 5 outlines the design of our video-on-demand system and describes the algorithms that are compared. Section 6 describes our simulator. Section 7 discusses our results. Section 8 briefly outlines how some additional features such as pause, rewind, and fast-forward might be added to the system. Finally, Section 9 presents some conclusions and future work.

2 Background

Any system using uncompressed digitized video data requires a tremendous amount of I/O bandwidth and storage capacity. For example, an uncompressed NTSC (the U.S. broadcast standard) video stream proceeds at about 30 Mbytes/second and a 90 minute uncompressed movie requires 150 Gbytes of storage. HDTV videos will require even more bandwidth and storage. Thus, most multimedia and video-on-demand systems will have to use some form of data compression to be both feasible and cost-effective.

This study assumes that all movies are compressed using the MPEG video compression standard [Gall91]. MPEG uses a lossy compression algorithm. In other words, a decompressed MPEG video stream will not be identical to the original video before compression. Although lossy compression is not acceptable for some applications (e.g., the storage of medical or scientific data where errors introduced during compression could have potentially serious consequences), the deterioration in picture quality is minimal and is not easily detected by an untrained viewer. Moreover, the MPEG compression ratio of 1:25 or higher is much better than the compression achieved by traditional lossless algorithms. Hence, MPEG compression is a good choice for a movie-on-demand system.

The MPEG-I standard recommends a "constrained parameter set" that results in a compressed bit stream of 1.5 Mbits/second. This particular bit rate was chosen because it corresponds to the CD-ROM bit rate. However, the

*This research was partially supported by ARPA under contract number DAAB07-92-C-Q508 and monitored by the US Army Research Laboratory This research was partially supported by NSF grant number CDA-9024618 This material is based on work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

SIGMOD ' 95,San Jose , CA USA
© 1995 ACM 0-89791-731-6/95/0005..$3.50

resulting uncompressed video is at best VHS quality and is certainly lower quality than broadcast television. Consumers will demand high-quality pictures from a video-on-demand service. As a result, this study assumes a 4 Mbits/second video stream with a resulting picture that is as good as or better than broadcast television.

Finally, a successful video-on-demand system cannot simply offer a small set of very popular movies; an enhanced pay-per-view system could provide such a service. Rather, a video-on-demand system must offer customers a large selection of videos. Just as a small set of movies account for a substantial percentage of all rentals at a video store, in a video-on-demand system some of the movies will be extremely popular and accessed frequently while others will be requested less frequently. Furthermore, the most frequently requested videos may change not only on a weekly or monthly basis, but even on an hourly or daily basis. For example, children's movies are likely to be popular early in the evening or on Saturday mornings, but less popular late at night. Thus, a video-on-demand system must be able to adapt rapidly to a widely varying and highly dynamic workload.

3 Related Work

[Özde94] proposes a movie-on-demand system in which each movie is stored on a separate disk and an unlimited number of subscribers can view each movie. Rowe [Fede94, Rowe92] is building a comprehensive system including a video file server, a continuous media player, and a large library of digitized videos. [Hsie94] studies the performance of an actual video server running on a Silicon Graphics Onyx computer system. [Suzu94] describes a system in which frequently referenced movies are stored on disk while infrequently referenced movies are stored in an optical jukebox. [Vin93] performs an analytical study of the design and operation of a video server including disk layout and admission control constraints. [Ande91] describes the design of a multimedia server, ACME, and presents a mechanism for synchronizing several media streams (e.g., audio and video). Other multimedia and continuous media systems include the Continuous Media File System [Ande92], the Continuous Media Storage Server [Loug92], and the IBM Shark Continuous Media File Server [Hask93].

[Yu92] presents a disk scheduling algorithm referred to as the group sweeping scheme (GSS). The intent of this algorithm is to minimize both the disk access time and the amount of buffer space. Section 7 of this paper includes results using GSS.

[Redd94] simulates a real-time I/O subsystem including a SCSI bus and disks and compares the performance of three disk scheduling algorithms, elevator, earliest-deadline-first (EDF), and a hybrid of the first two, with varying amounts of memory. [Hari94] presents an analytical and simulation study comparing the memory requirements of the elevator and first-come-first-served (FCFS) disk scheduling algorithms.

[Bers94] introduces a file layout technique called staggered striping that is suitable for the storage of high-bandwidth video objects that may not be compressed such as medical or scientific data. [Chan94] compares various strategies for allocating video data on disk arrays such that the video can be efficiently retrieved at differing quality levels. [Chen93] uses an analytical study to develop a disk layout strategy for multiple related multimedia streams such as video, audio, and text (e.g., closed captioning). [Tetz94] uses an analytical study to discuss the effects of various parameters and technology trends on a video-on-demand system.

4 Our Work

This paper simulates a variety of video server hardware configurations and algorithms to determine how such a server should be designed and configured to minimize cost and maximize performance. Many other video-on-demand systems have been designed by starting with an analytical performance study. Although a system based on an analytical study can, at least in theory, guarantee a certain level of performance (e.g., no glitches), often analytical studies make worst case assumptions (e.g., maximum disk seeks and latencies). Furthermore, many systems designed around analytical studies necessarily must assume overly simplistic algorithms (e.g., round-robin disk scheduling). Thus, such a system may be over-designed or pessimistic and may not achieve the maximum possible utilization of the hardware.

By instead using a detailed simulation model, the performance of a system can be accurately predicted using realistic disk, network, and CPU models. The maximum number of simultaneous users that each configuration of the video server is able to support can be determined and the effects of varying each of the design parameters can be accurately measured.

Our primary metric for the performance of a particular system configuration is the maximum number of users that the configuration can support while providing continuous glitch-free video. This method of evaluating a video-on-demand system is compatible with the method taken in most analytical studies. The only difference is that most analytical studies begin with the assumption that users should never experience glitches and proceed to design a system around that assumption while we first design a system and then analyze whether that design resulted in glitches.

Finally, this study does not attempt to prove that the simulated systems will never produce a glitch. As we indicated above, a system that is designed around an analytical study and is proven never to cause a glitch, is unlikely to achieve high utilization of the hardware. Furthermore, in any real system, unexpected and uncontrollable events can always cause glitches. However, the risk of glitches can be made arbitrarily low by limiting the maximum number of terminals as much as is desired.

5 The SPIFFI Video-on-Demand System

Figure 1 provides an overview of the SPIFFI video-on-demand system hardware. The video server itself consists of a set of nodes connected by a very high-speed interconnection network such as ATM. This study does not address the details

of the network and assumes that it is not a bottleneck. Each node consists of a CPU, some memory for prefetching and buffering video data, a network interface, and a set of disks. The video server's clients are a set of video terminals with real-time decompression hardware and memory for local buffering of video data and are attached to the same high-speed interconnection network.

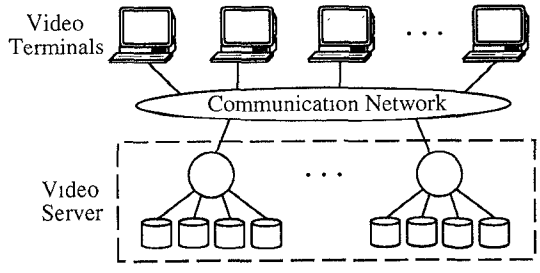


Figure 1: Overview of the SPIFFI video-on-demand system.

The SPIFFI video-on-demand system is designed to be built out of inexpensive commodity components (e.g., PCs or workstations with SCSI disks and plug-in network devices). This approach differs from the one taken by some vendors [Laur94] who are building systems based around costly massively parallel processors. Our results in Section 7 show that it is not necessary to spend a great deal of money to build a viable video-on-demand system. In the following sections, the video terminals and video server are discussed in detail.

5.1 The Video Terminals

The video terminals are used by movie-on-demand subscribers to select and view movies. It neither possible nor desirable for a terminal to retrieve a video in its entirety before beginning display of it. Instead each terminal attempts to buffer only as much of the video as necessary to ensure that display of the movie can proceed uninterrupted while the next block of video is retrieved from the server.

Before initiating display of a movie, a terminal first fills or primes its buffers with video data. Then, as illustrated by Figure 2, it begins decompressing and displaying the movie while simultaneously retrieving subsequent blocks of video. A terminal will always request more video data from the video server as long as it has the memory to buffer it. If the terminal completely fills its buffer space, it temporarily suspends the retrieval of additional video data until some of the buffered data is displayed. If the terminal runs out of video to display, a "glitch" occurs and the terminal must pause the movie while it waits for more data to arrive. If a glitch does occur, the terminal re-primes its buffers before restarting display of the video. This strategy increases the duration of the glitch but reduces the likelihood of a second glitch occuring immediately after the first.

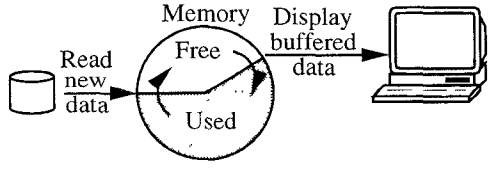


Figure 2: Video terminal operation.

5.2 The Video Server

The video server software is based on SPIFFI, a high-performance, scalable parallel file system that we developed for and implemented on an Intel Paragon [Free94]. SPIFFI provides much of the necessary I/O functionality needed to build a video server. Some of the features that were particularly important were file striping, decentralized file accesses, and buffering and prefetching.

SPIFFI automatically stripes files across all the disks in the video server [Ries78]. As Figure 3 illustrates, when SPIFFI declusters a file, it first alternates between the nodes and then between the disks at each node. Thus, block A.0 is stored on node 0, disk 0; block A.1 is stored on node 1, disk 0; block A.2 is stored on node 0, disk 1; etc. The portion of a video stored on one disk (e.g., blocks B.3, B.7, B.11, etc.) is called a **fragment** and is laid out contiguously. Each block (e.g., A.3) is referred to as a **stripe block**. The size of each stripe block is constant and is termed the **stripe size**.

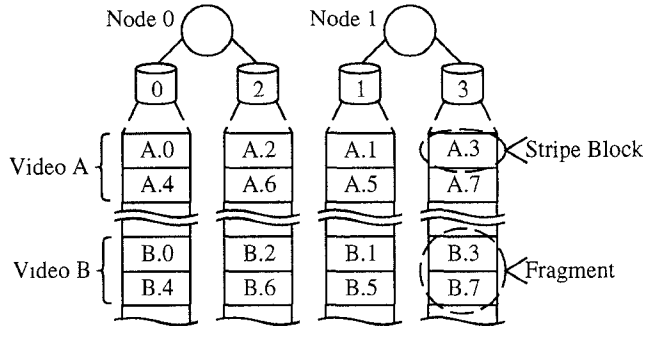


Figure 3: Striping of videos across multiple disks.

Fully striping the videos has two advantages. First, the entire aggregate bandwidth of all the disks in the system is available to show the most popular movies. This advantage is particularly important since, as discussed in Section 2, the movies most in demand are likely to change during each 24 hour period as well as on a daily and weekly basis. By fully striping all videos, the system can automatically adapt to workload changes without having to reorganize the disk layout of the videos. Second, unpopular movies do not render the disks that store them underutilized. That is, striping automatically balances the I/O load across all the server's disks ensuring that no disks are either under or over utilized.

SPIFFI's decentralized implementation streamlines the performance and cost of the video server by routing read requests for video data directly from the terminal to the appropriate node and disk. Read requests need not pass through any intermediate nodes and there is no need to consult a global page mapping database before each disk access.

Finally, SPIFFI provides a buffer pool that includes a highly effective prefetching mechanism. The basic SPIFFI buffer pool uses a **global LRU** (least recently used) page replacement algorithm and the **elevator** disk scheduling algorithm that balances minimizing seek times against

fairness [Silb94]. After servicing an I/O request for a terminal, SPIFFI issues a background prefetch request for the next stripe block of the video file. When the terminal ultimately requests this block, if the prefetch has completed, the request will be satisfied by the buffer pool with a significant reduction in service time. By prefetching, SPIFFI takes advantage of idle time at the disks to do useful work.

As part of this project, we enhanced the SPIFFI buffer pool by adding a new page replacement algorithm, **love prefetch**, that favors prefetched pages over already referenced pages [Teng84]. We augmented the disk scheduler with a **real-time** disk scheduling algorithm and with an implementation of the **group sweeping scheme (GSS)** proposed in [Yu92]. Finally, we added two new prefetching strategies, **real-time prefetching** and **delayed prefetching**. The details of these algorithms is described in the following sections.

5.2.1 Buffer Pool Page Replacement Algorithms

The **global LRU** algorithm [Silb94] simply places newly referenced pages (i.e., stripe blocks) onto the end of a single queue. When a new page is needed, the buffer pool searches for the first available page starting from the head of the queue. This algorithm does not distinguish between **prefetched pages** and **referenced pages** (i.e., pages or stripe blocks that have been explicitly requested by a video terminal).

Due to the huge size of the video files (2 hours equals 4 Gbytes) and the strictly sequential access pattern, it is impossible to cache a significant portion of a video in memory for reuse and the likelihood that a stripe block in the buffer pool will be referenced more than once is low [Özde94]. The **love prefetch** page replacement algorithm takes advantage of this fact and breaks the global LRU chain into two separate LRU chains: one for prefetched pages and one for referenced pages [Teng84]. As Figure 4 illustrates, when a stripe block is first prefetched, it is placed on the prefetched-pages LRU chain. When it is subsequently referenced, it is moved to the referenced-pages LRU chain. When a new page is needed, the buffer pool first attempts to find an available page on the referenced-pages LRU chain. If there are no available pages on the referenced-pages LRU chain, the buffer pool takes a page from the prefetched-pages LRU chain.

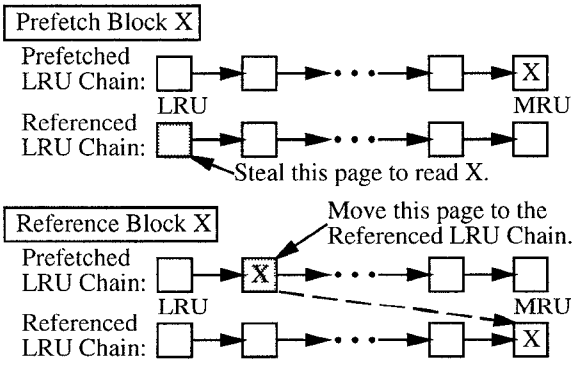


Figure 4: The love prefetch page replacement algorithm.

5.2.2 Disk Scheduling Algorithms

The **elevator** disk scheduling algorithm [Silb94] scans the disk cylinders starting with the innermost cylinder and working outward. When it reaches the outermost cylinder, the algorithm reverses and begins scanning inward. An I/O request is serviced when the disk head reaches its cylinder. This algorithm is popular because it combines nearly minimal seek times and fairness.

The **real-time** disk scheduling algorithm is an extension of an existing **priority** disk scheduling algorithm which is itself an extension of the elevator algorithm [Care89]. First, terminals assign each disk access a deadline by which it must be completed to avoid a glitch. This deadline is used to assign each pending disk access to one of a fixed set of priority classes. There are many ways to map deadlines into priorities. To reduce the number of possible mappings, we use uniformly spaced **priority cutoffs**. For example, in Figure 5, there are 3 **priority classes** or simply 3 **priorities** and the spacing between priority cutoffs or the **priority spacing** is 2 seconds. Thus, the priority cutoffs are at 2 seconds and 4 seconds. Those requests within 2 seconds of their deadlines are assigned to the highest priority class while those requests with more than 4 seconds remaining are assigned to the lowest.

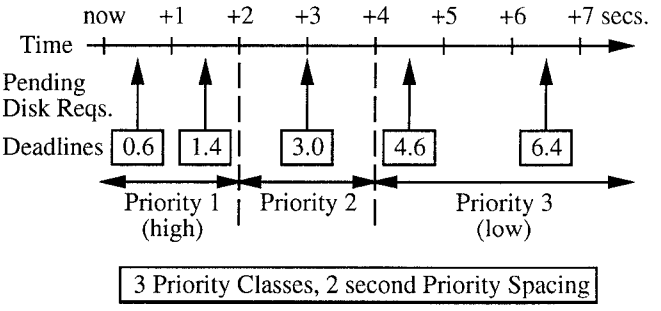


Figure 5: Assigning priorities to disk accesses.

After all disk accesses are assigned a priority, the highest priority class with pending disk accesses is selected and serviced using the elevator algorithm. For instance, in Figure 6 request 2 is assigned priority 1 and is serviced first even though the disk head must seek past cylinder 10 and skip request 1 which only has priority 2. After each disk access, priorities are recomputed using the current time. Thus, continuing the example, request 1 is now within 2 seconds of its deadline, is promoted to priority 1, and is serviced next.

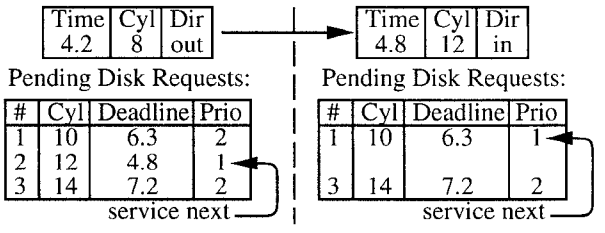


Figure 6: The real-time disk scheduling algorithm.

For comparison, we also implemented the **group sweeping scheme (GSS)** described in [Yu92]. This algorithm assigns each terminal to one of a fixed set of groups. These groups are processed repeatedly in round-robin order. To process a group, up to one request from each terminal within that group is selected and serviced using the elevator algorithm. Increasing the number of groups (and assigning a small number of terminals to each group) causes the terminals to be serviced more or less in round-robin order, thus, reducing the maximum time that a terminal may have to wait to be serviced and the maximum amount of video data that each terminal must buffer to avoid a glitch. Decreasing the number of groups (and assigning many terminals to each group) causes the terminals to be serviced more or less in elevator order which increases the efficiency of the disk scheduling but also increases the maximum time that a terminal may have to wait to be serviced. At one extreme, with one group the algorithm is nearly identical to the elevator algorithm. The only difference is that with GSS each terminal is serviced at most once on each pass over the disk while with elevator a terminal may be serviced many times on each pass. At the other extreme, if the number of groups is equal to the number of terminals, the algorithm is simply **round-robin**.

5.2.3 Prefetching Algorithms

The standard SPIFFI prefetching algorithm operates by responding to each real reference to a stripe block on some disk with a background request for the next stripe block at the same disk. Each prefetch request is inserted into a first-in first-out queue associated with the appropriate disk. A fixed set of prefetch processes service each disk's prefetch queue. Once a prefetch request has been issued to the disk, it is serviced according to the disk scheduling policy in effect. If the real-time algorithm is used, prefetch requests are assigned by default to the lowest possible priority.

By varying the number of prefetch processes and, hence, the number of prefetch requests that are concurrently in the disk queue, the "aggressiveness" of the prefetching mechanism can be altered. Using many prefetch processes leads to more aggressive prefetching while using fewer processes leads to less aggressive prefetching. The non-real-time disk scheduling algorithms are hurt by aggressive prefetching because they do not distinguish between an urgent request issued by a terminal and a non-urgent prefetch request. The real-time disk scheduling algorithm can identify and skip prefetches if necessary and, therefore, benefits from aggressive prefetching. In each experiment, the prefetching mechanism was configured to maximize the performance of the disk scheduling algorithm in use.

We extended the basic prefetching algorithm in two ways. The **real-time prefetching** algorithm is designed to augment the real-time disk scheduling algorithm described above. For each prefetch request it estimates the deadline that will ultimately be assigned to the anticipated true request for the same stripe block. Requests are inserted into a priority queue so that the request with the most urgent deadline will always be at the head of the queue. Finally, the real-time disk scheduling algorithm uses a prefetch request's deadline to assign it a priority just as it assigns priorities to real requests. Thus, an urgent prefetch request can take priority over a non-urgent true request. This algorithm always benefits the real-time disk scheduling algorithm and, therefore, these two algorithms are always used together.

The **delayed prefetching** algorithm is a further extension of the real-time prefetching algorithm and is intended to reduce the overall memory requirements of the server. Often, prefetch requests are issued and serviced several seconds before the data is really needed. As a result, extra memory is required to buffer the results of the prefetch until the actual request for the data arrives. As Figure 7 shows, the delayed prefetching algorithm simply delays issuing a prefetch until it has less than a set amount of time left before its deadline. The precise amount of time between a prefetch's deadline and when it may be issued is referred to as the **maximum advance prefetch time**.

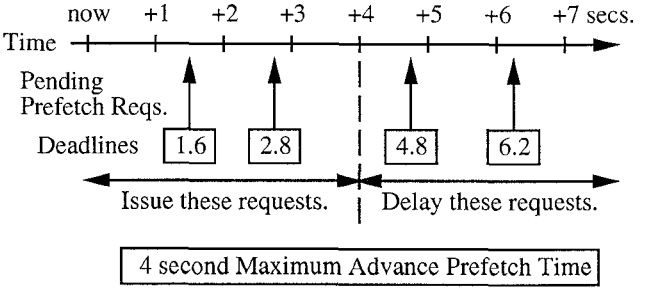


Figure 7: The delayed prefetching algorithm.

6 Simulator Description

Our video-on-demand simulator is based on an existing SPIFFI simulator that was validated against an actual SPIFFI implementation on a 64 node/64 disk Intel Paragon [Free94]. The event-driven simulator is written in the CSIM/C++ process-oriented simulation language [Schw90].

We are primarily interested in the performance of the system when all the terminals are actively viewing movies. Furthermore, in a real system videos will be started at random intervals. To capture this behavior, when a simulation begins, the terminals start movies at random intervals. When a terminal finishes one movie, it randomly selects a new video and immediately begins playing it. Once all the terminals have begun watching videos, the simulator begins collecting performance and utilization data. The simulation continues for a fixed period of simulated time and then is terminated abruptly. Thus, the simulation is of a closed system of terminals and the results represent a snapshot of the system's performance with all the terminals active.

The simulator parameters are summarized in Table 1. In the remainder of this section, the various components of the simulator are described in more detail.

6.1 Videos

To make the simulator as accurate as possible, the display of individual MPEG frames is simulated. An MPEG

Configuration Parameter	Value
MPEG I:P:B frame freq ratio	1:4:10
MPEG I:P:B frame size ratio	10:5:2
Video Bit Rate	4 Mbits/second
Video Length	60 minutes
Number of CPUs	4
Number of Disks/CPU	4 to 16
CPU Speed	40 MIPS
CPU Scheduling	FCFS
Disk Seek Factor	0.283
Disk Settle Time	0.75 ms
Disk Rotation Time	8.333 ms
Disk Transfer Rate	7.4 Mbyte/sec
Disk Cache Context Size	128 Kbytes
Disk Cache Size	8 contexts
Disk Cylinder Size	1.25 Mbytes
Network Wire Delay	5 μs + 0.04 μs/byte
Start an I/O	20000 instrs[1]
Send a message	6800 instrs
Receive a message	2200 instrs

Table 1: Simulator parameters.

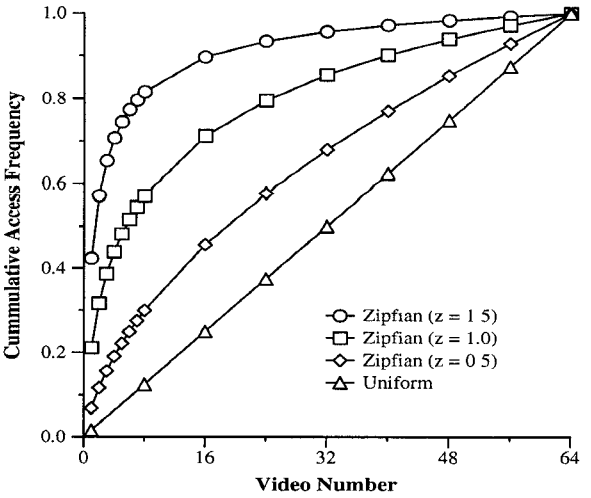


Figure 8: Zipfian distribution.

compressed video stream contains three types of frames: intra (I), predicted (P), and bidirectional (B). I frames are the least common and largest frame type while B frames are the most common and smallest frame type. For this study, the I:P:B frame frequency ratio was 1:4:10 and the I:P:B frame size ratio was 10:5:2 with an overall frame rate of 4 Mbits/second (NTSC broadcast quality) [Gall91]. The specific sizes of frames of the same type vary depending on the contents of the uncompressed frames. An analysis of several MPEG videos showed that frame sizes typically are exponentially distributed. Thus, each simulated video consists of a sequence of I, P, and B frames with exponentially distributed sizes. Each time the same video is played, the same sequence of frames and frame sizes is repeated.

The simulated video library consists of 4 one hour long videos per disk. Thus, a configuration with 16 disks would have 64 videos with each video striped across all 16 disks. The selection of one hour for the video length was intended to limit the simulation time[2] while remaining reasonably representative of true movies. The number of videos was limited by drive capacity (4 one hour videos per disk equates to about 7 Gbytes per disk).

As discussed in Section 2, in any video-on-demand system certain popular movies will be accessed frequently while other less popular selections will be requested less frequently. This access pattern corresponds to a Zipfian distribution (see Figure 8) [Zipf49]. The parameter, z, determines how skewed the distribution is. For most of our experiments, except where otherwise noted, z was set to 1.

6.2 Disks and Network

The disks are based on the Seagate ST15150N SCSI-2 disk (5 Gbyte capacity) which is currently a state-of-the-art high-performance model. Although this disk has variable capacity cylinders, for simplicity and ease of implementation a constant cylinder size is assumed. No other simplifying assumptions are made about this drive.

As indicated in Section 5, the details of the network design are not considered as part of this study and the network is assumed not to be a bottleneck. Thus, the network is modeled as a bus with unlimited aggregate bandwidth and constant latency regardless of which terminal and node are communicating. The CPU times to initiate send and receive operations as well as an appropriate wire delay based on the length of the message are all simulated. If necessary, messages are stored in a queue at the recipient's node or terminal while awaiting processing.

7 Results

In this section, we illustrate the effects of varying each of the video server parameters and compare the performance of the algorithms described in Section 5.2. The intent of this section is not to demonstrate what the optimal values of the video server parameters are since the optimal settings for many of the parameters (e.g., stripe size or amount of memory) ultimately depend on the exact hardware in use. Rather, our goal is to show what the effect is of varying these values or of using the various algorithms. These results can be used to improve the overall design of an actual video-on-demand system or to assist in its fine tuning.

To reduce the complexity of the analysis, we initially experiment only with 16 disks (4 processors with 4 disks per processor) and 64 videos (each video is one hour long). The amount of memory at the video server is fixed at 4 Gbytes (1 Gbyte at each of the 4 processors) and the basic global LRU page replacement algorithm is used. The amount of memory at the terminals is fixed at 2 Mbytes (enough to buffer approximately 4 seconds of video). After analyzing the effects of varying the stripe size and comparing the disk scheduling algorithms, we show that by using the alternative buffer pool page replacement and prefetching algorithms the amount of video server memory can be substantially reduced

[1] This value was measured on an Intel Paragon. Although it is high, the video server is still completely I/O bound.

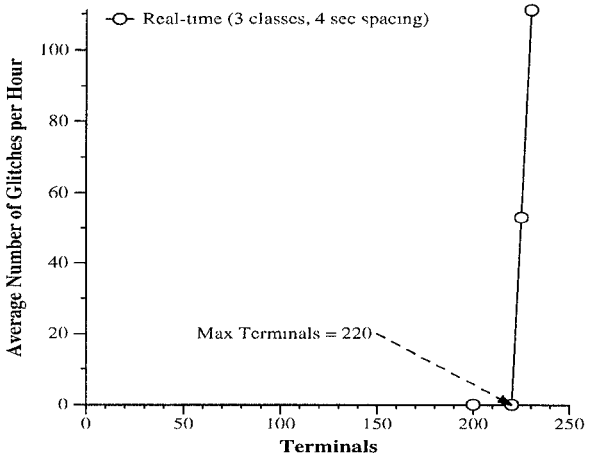
[2] A Sun SPARCstation 10 required up to 2.5 hours to run a 16 disk simulation, up to 5 hours to run a 32 disk simulation, and up to 10 hours to run a 64 disk simulation.

from 4 Gbytes without affecting the performance of the system. We also demonstrate that striping videos across the disks leads to substantially higher performance when compared with a non-striped configuration and investigate the effect of varying the relative access frequencies of the movies. We finish by presenting some scaleup results using 32 and 64 disks.

In all our experiments, the read request size at both the terminals and the server is set to the disk stripe size. Furthermore, the terminals carefully align read requests so that they correspond to exactly one stripe block (see Figure 3) and may always be serviced by a single disk. This strategy optimizes the performance of the system. Since a stripe block represents a contiguous portion of a video file, it will be completely consumed within a short period of time. The data just preceding and just following a stripe block belong to different portions of the video, are needed either much earlier or much later, and are, thus, read separately. Finally, by directing each read to a single disk, the terminals ensure that the completion time for each request is determined by a single disk and is, hence, minimal.

7.1 Metrics and Methodology

As discussed in Section 4, our primary metric is the maximum number of terminals that a configuration can support without glitches. This value is obtained by increasing the number of terminals until the number of glitches becomes non-zero. Figure 9 provides an example of this procedure. For the configuration illustrated in the graph the maximum number of terminals that can be serviced glitch-free is 220. To ensure that our results are accurate, we ran each experiment until we were 90% confident that the results were within 5% (about 10 terminals) of the actual maximum number of terminals.



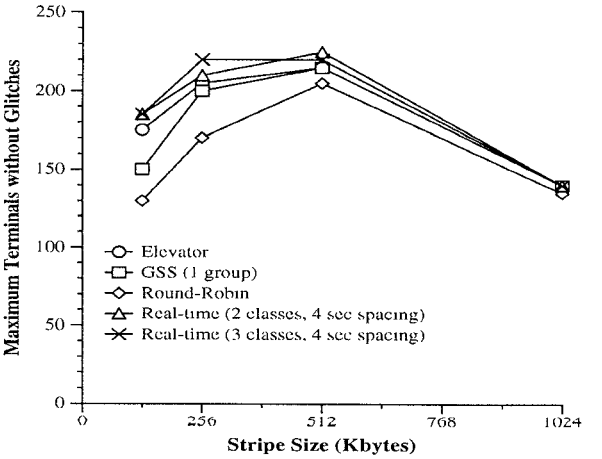
512 KB Stripe Size, 4 GB Memory, Global-LRU

Figure 9: Finding the max. no. of terminals without glitches.

7.2 Disk Scheduling Algs. and Stripe Sizes

Our first experiment compares the performance of the various disk scheduling algorithms over a range of stripe sizes. As indicated at the beginning of this section, we simulated a 4 processor, 16 disk (4 disks per processor) system with 4 Gbytes of server memory (1 Gbyte per processor; enough to ensure that memory does not limit performance) and the global LRU page replacement algorithm. Each terminal had 2 Mbytes of memory (enough to buffer 4 seconds of video). The stripe size was varied from 128 to 1024 Kbytes and the elevator, one group GSS, round-robin, and real-time disk scheduling algorithms were compared. Although the real-time disk scheduling algorithm takes two parameters (the number of priority classes and the priority spacing) and, hence, has numerous variations, only two are shown: 2 priority classes with 4 second priority spacing and 3 priority classes with 4 seconds priority spacing. We explored a wide variety of settings for these parameters and found that regardless of how they were set there was little variation in the performance of the system.

Figure 10 compares the various disk scheduling algorithms over a variety of stripe sizes. These results show that among the tested configurations elevator and both real-time disk scheduling algorithms perform nearly identically at all stripe sizes. The best performance of 225 terminals is achieved with real-time disk scheduling and a stripe size of 512 Kbytes. As the stripe size decreases, the disks spend more time seeking per byte read and, hence, performance slowly declines. With a stripe size of 1024 Kbytes, performance drops substantially because the time to complete each disk read is too long relative to the amount of video buffered at the terminals. Thus, choosing a stripe size is a balance between minimizing seeks, I/O service times, and video terminal memory. Figure 10 also shows that while one group GSS works well with a stripe size of 512 Kbytes, it performs more poorly as the stripe size decreases. Round-robin always performs more poorly than the other disk scheduling algorithms as it makes no attempts to optimize seek distances.



4 GB Aggregate Server Memory, Global-LRU

Figure 10: Comparison of disk sched. algs. and stripe sizes.

7.3 Video Server Memory Requirements

In the previous section, the subject of the server's memory requirements was ignored by using 4 Gbytes of memory and the basic global LRU page replacement algorithm. In this experiment, the aggregate amount of memory at the server

was reduced to as little as 128 Mbytes (with fewer than 128 Mbytes the server began to run out of free pages) and the performance of the global LRU, love prefetch, and delayed prefetching algorithms were compared. Terminals continued to have 2 Mbytes of memory for buffering video.

This experiment was repeated once with elevator disk scheduling and again with real-time scheduling with 3 priority classes and 4 second priority spacing. The stripe size was fixed at 512 Kbytes which was shown to optimal in the preceding section. With elevator disk scheduling only global LRU and love prefetch were compared since delayed prefetching can only be used in conjunction with an algorithm such as real-time scheduling that assigns deadlines to requests. With real-time scheduling global LRU, love prefetch, and delayed prefetching were all compared. When delayed prefetching was used, its parameter, the maximum advance prefetch time, was set to 8 and 4 seconds. As discussed in Section 5.2.3, this parameter determines how far in advance of its estimated deadline a stripe block may be prefetched.

Figure 11 presents the results using elevator disk scheduling. With global LRU the performance of the system declines when the server has less than 512 Mbytes. When the love prefetch algorithm is used the system continues to work well with as little as 128 Mbytes. Figure 12 presents the results using real-time disk scheduling. In this case, global LRU performs extremely poorly as soon as the amount of memory is reduced below 4 Gbytes. With the love prefetch algorithm but unconstrained prefetching, performance declines with less than 1 Gbyte. The love prefetch and delayed prefetching (8 seconds) combination functions well with as little as 512 Mbytes. Finally, when delayed prefetching is used with only a 4 second maximum advance prefetch time the performance of the system is significantly worse (30 to 40 fewer terminals) regardless of how much memory is used. Thus, at current memory prices of about $40/Mbyte, the love prefetch and delayed prefetching algorithms can reduce the cost of the server memory to as little as $5,000 to $20,000.

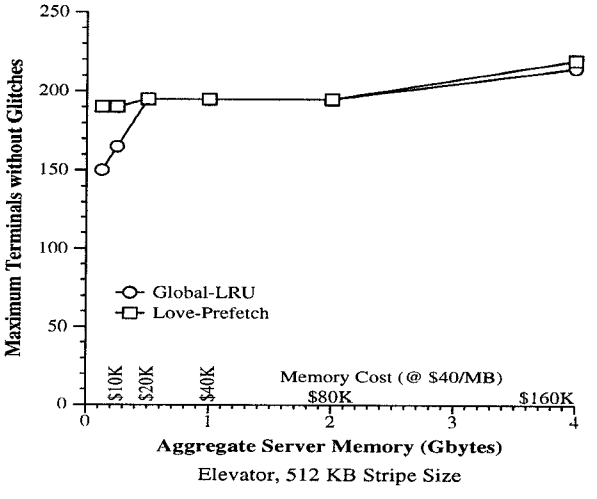


Figure 11: Reducing server memory requirements.

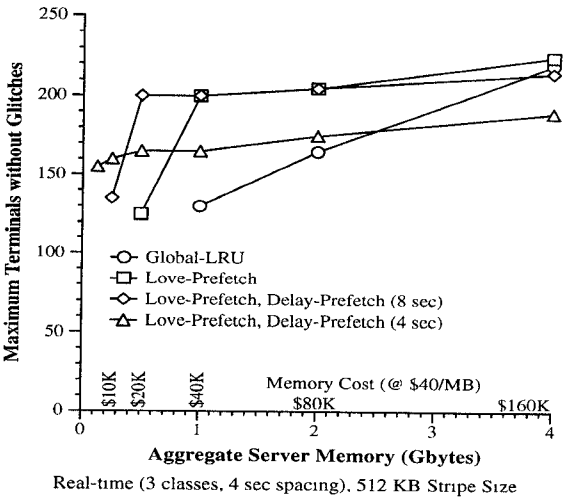


Figure 12: Reducing server memory requirements.

The reason that global LRU performs much more poorly with real-time disk scheduling than with elevator is that with real-time scheduling, the server attempts to prefetch aggressively, using the real-time prefetching algorithm described in Section 5.2.3, while with elevator, prefetching is severely limited to avoid interfering with actual I/O requests from the terminals. Aggressive prefetching requires extra memory in which to store the prefetched pages. If a prefetched page cannot be cached until it is referenced, the prefetch will have been a waste and the page will have to be read a second time when it is ultimately referenced. The love prefetch and delayed prefetching algorithms prevent wasted prefetches and, consequently, reduce the amount of memory required at the server. However, when delayed prefetching (4 seconds) is used (Figure 12), performance begins to suffer because prefetches are issued too late to prevent actual terminal requests from resulting in disk I/Os.

7.4 Non-Striped Disk Layout

In Section 5.2 we asserted that striping videos is necessary to achieve maximum utilization of the disks and to adjust dynamically to changing movie access patterns. In this experiment this assertion is validated by comparing the performance of a system with fully striped videos to a system with non-striped videos. In the case of the non-striped videos, each video was stored on a single, randomly chosen disk and each disk held exactly 4 videos. For the striped case, a stripe size of 512 Kbytes was used. A read size of 512 Kbytes was used for both the striped and non-striped configurations. Love prefetch page replacement and elevator disk scheduling were used in both cases. The aggregate amount of server memory was varied from 128 Mbytes to 4 Gbytes.

As Figure 13 shows, if the videos are requested using a Zipfian distribution, the performance using the non-striped layout is extremely poor (only 30 terminals). The problem is that those disks storing the more popular movies rapidly become overloaded while the other disks remain virtually idle. Therefore, for comparison, a uniform access pattern was also simulated. However, even with the uniform distribution,

the non-striped layout leads to far worse performance (still only 80 terminals) than the striped layout which supports about 190 terminals regardless of whether videos are accessed using a uniform or Zipfian distribution. Again, the reason is that some disks experience many more requests than others. Although the access pattern is uniform, there is still enough variation to overload some disks while others remain largely underutilized. Figure 14, which compares the average disk utilization for the striped and non-striped layouts, shows the extent of the underutilization. With the non-striped layouts, average disk utilization never exceeds about 40% while with the fully striped layout disk utilization approaches 100%.

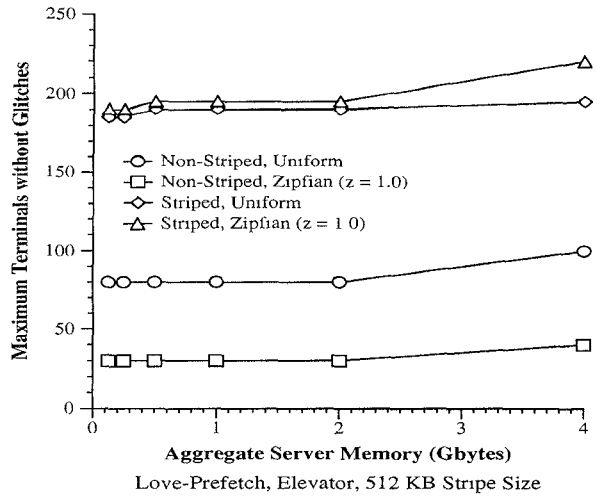


Figure 13: Striped vs. non-striped layouts.

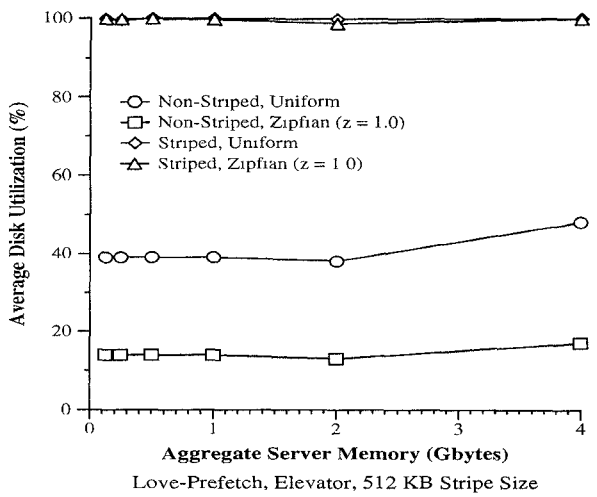


Figure 14: Striped vs. non-striped layouts.

7.5 Movie Access Frequencies

As discussed in Sections 2 and 6.1, some movies are likely to be accessed much more frequently than others. Thus, throughout this study we have assumed that terminals request movies using a Zipfian distribution. In this experiment, the access frequencies of the movies was varied. A uniform distribution and three Zipfian distributions with z equal to 0.5, 1.0, and 1.5 (see Figure 8) were compared. As in the previous experiment, a stripe size of 512 Kbytes, love prefetch page replacement, and elevator disk scheduling were used. Again, the aggregate amount of server memory was varied from 128 Mbytes to 4 Gbytes.

Figure 15 shows that, with very little memory, performance is independent of the video access frequencies. However, as the amount of memory increases, the more skewed distributions (Zipfian with z equal to 1.0 and 1.5) outperform the less skewed distributions. As the access frequencies become more skewed, the probability increases that two terminals will independently choose to watch the same video at approximately the same time. Thus, there is an increase in the number of stripe blocks that are referenced by more than one terminal. This increase is clearly visible in Figure 16 which shows the percentage of buffer pool references that request a page (i.e., stripe block) that was previously referenced by another terminal.

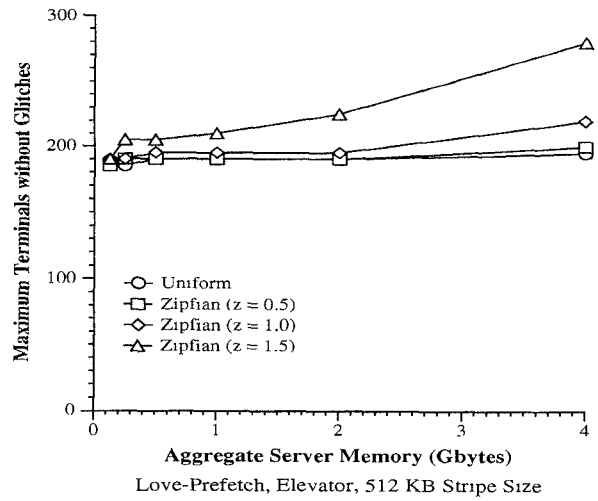


Figure 15: Varying movie access frequencies.

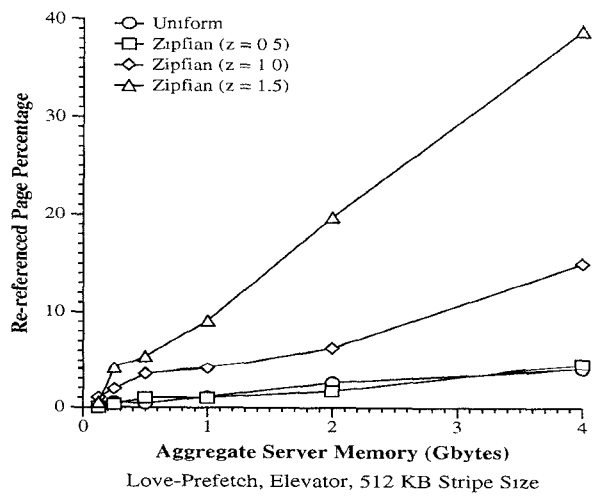


Figure 16: Varying movie access frequencies.

7.6 Scaleup

Our final experiment is intended to show that our video-on-demand system continues to perform well even as the number

Disk Scheduling	Terminal Mbytes	Base			×2				×4			
		Disks	Server Mbytes	Max. Terms.	Disks	Server Mbytes	Max. Terms.		Disks	Server Mbytes	Max. Terms.	
Elevator	2	16	128	190	32	256	345	(0.91)	64	512	535	(0.70)
Elevator	2.5	16	128	195	32	256	385	(0.99)	64	512	750	(0.96)
Elevator	2	16	512	195	32	1024	380	(0.97)	64	2048	620	(0.79)
Real-time	2	16	512	200	32	1024	395	(0.99)	64	2048	760	(0.95)

Table 2: Scaleup.

of disks and terminals is increased. This experiment was repeated with the four different base server configurations. All four base configurations used 4 processors, 16 disks, 64 videos, and a 512 Kbyte stripe size. The first configuration used elevator disk scheduling with 128 Mbytes of server memory, love prefetch page replacement, and 2 Mbytes of memory per terminal. The second configuration was the same except that the amount of terminal memory was increased to 2.5 Mbytes. The third configuration was also the same as the first except that the server memory was raised to 512 Mbytes. Finally, the fourth configuration used real-time disk scheduling (3 priority classes and 4 second priority spacing) with 512 Mbytes of server memory, love prefetch page replacement, delayed prefetching (8 seconds), and 2 Mbytes of terminal memory. The first and fourth configurations were found in the preceding sections to be optimal for the elevator and real-time disk scheduling algorithms respectively. The second and third configurations were chosen for comparison purposes. As the number of disks was increased from 16 to 32 to 64, the amount of server memory and the number of videos were also increased proportionately. Four CPUs were used regardless of the number of disks. All other parameters were also left unchanged as the number of disks was increased.

Table 2, which presents the scaleup results for each of the above configurations, shows that elevator disk scheduling does not scale well. To maintain good performance as the size of the server is increased, the elevator algorithm requires that the amount of memory at the terminals also be increased. This problem arises because as the number of terminals increases, the maximum queue length and, hence, the worst-case service time at the disks slowly increases. With the elevator algorithm the amount of video buffered at the terminals must be increased to compensate for the increased service times. The real-time algorithm is able to schedule disk requests based on when they are needed and, therefore, does not suffer from this problem. It scales nearly linearly to at least 64 disks, 256 videos, and 760 terminals.

As Figure 17 shows, CPU utilization is not a performance factor even with the 16 disks per node (64 disks total) configuration. Even if the CPU utilization, memory bandwidth, or I/O bus was to become a bottleneck, the number of nodes and CPUs could easily be increased and the limitation overcome due to the "shared-nothing" design of the system. Figure 18 shows the peak aggregate network bandwidth used by the system as it is scaled. For instance,

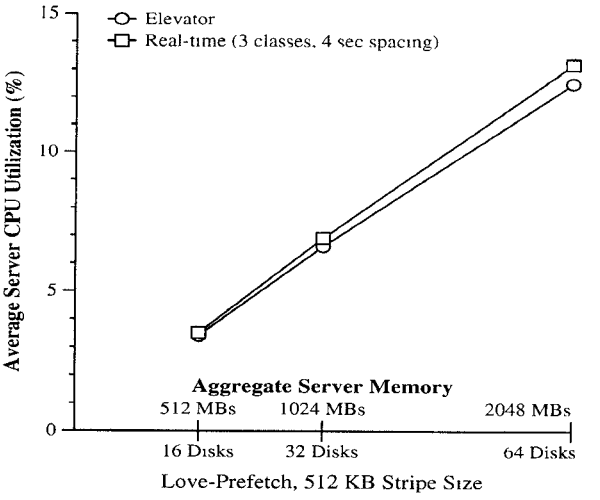


Figure 17: CPU utilization.

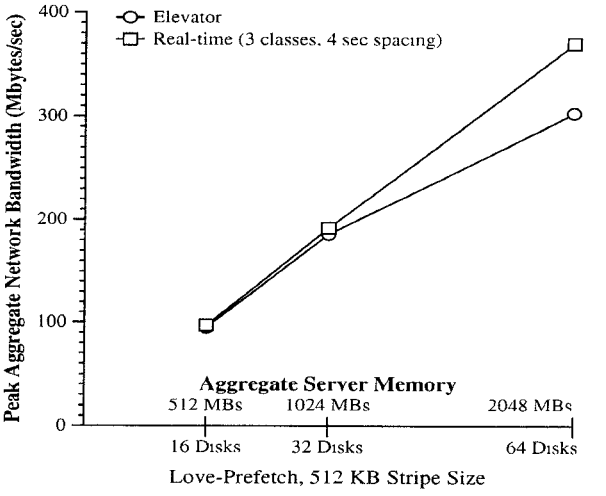


Figure 18: Peak aggr. network bandwidth reqs.

with 64 disks and 760 terminals, the system requires an aggregate network bandwidth of just over 370 Mbytes/second or about 4 Mbits/second per terminal (the compressed video bit rate). Finally, disk utilization remains above 95% as the system is scaled.

As part of this study, we considered the disk-memory tradeoff and attempted to find a video server analogy to Gray's five minute rule for database systems [Gray93]. This rule states that database systems can achieve optimal price-performance by keeping those objects in memory which are accessed at least once every five minutes. However,

as mentioned briefly in Section 5.2.1, due to the large size of the videos (2 hours equals 4 Gbytes) and the sequential access pattern, caching video data for reuse is impossible. Thus, there is no analogy to the five minute rule for video servers, and, as the results in Section 7.3 and in this section confirm, it is best to purchase the minimum amount of memory necessary (as extra memory has a minimal impact on performance) and spend additional money purchasing extra disks which do significantly increases the number of terminals that may be supported without glitches.

At the time of this writing, 9 Gbyte disks are available for approximately $4,000, and memory costs about $40/Mbyte. Thus, the total storage (disk and memory) costs for a 64 disk, 2 Gbyte video-on-demand server would run about $338,000 or about $445 per "active" terminal.

Finally, in this experiment, the size of the video server was scaled by increasing both the number of disks and the number of videos. However, as disks increase in capacity, it might be tempting to scale a video server by switching to higher capacity disks rather than by increasing the number of disks. Although using higher capacity disks may result in a lower storage cost per Mbyte, they are unlikely to increase the maximum number of concurrent users that a video server supports glitch-free. Thus, there is a cost-performance tradeoff that must be considered when a video-on-demand server is designed. For example, a system to support 64 videos could be built using 16 9-Gbyte, 32 4.5-Gbyte, or 64 2.2-Gbyte drives. Table 3 combines current disk prices with the results in this section to compare the disk cost per terminal and illustrates that minimizing a system's cost per Mbyte does not lead to a minimal cost per terminal.

Videos	64	64	64
Disks	16	32	64
Capacity	9 GBs	4.5 GBs	2.2 GBs
Cost/Disk	$4,000	$2,500	$1,500
Cost/Mbyte	$0.43	$0.54	$0.67
Total Cost	$64,000	$80,000	$ 96,000
Terminals	200	395	760
Cost/Terminal	$320	$200	$125

Table 3: Comparison of disk costs per terminal.

8 Additional Features and Functionality

This section addresses how the SPIFFI video-on-demand system might provide extra features that have not yet been discussed. We explore how functions such as pause, rewind, and fast-forward may be implemented. We also consider how terminals viewing the same movie might be "piggybacked" to minimize the demand placed on the video server.

8.1 Supporting Pause, Rewind, and Fast-Forward

Until now, only the sequential playback of movies has been considered. A video-on-demand system must also support pause and restart, rewind, and fast-forward. The SPIFFI video-on-demand system is capable of easily providing this support. The video server makes no special assumptions about what portion of a particular video is currently being viewed at any particular terminal. Thus, for example, if a user presses pause, the terminal can simply halt the display of video. Then, when the user presses play again, the terminal can simply begin playing from where it left off. (It can even use the time during which it is paused to fill its buffers and prepare for the restart.) The procedure for the terminal is the same regardless of where in the video it begins playback.

Figure 19 shows the results of an experiment in which each terminal paused each video on average twice for an average of 2 minutes. As can easily be seen from the graph, performance is essentially unaffected by the pausing.

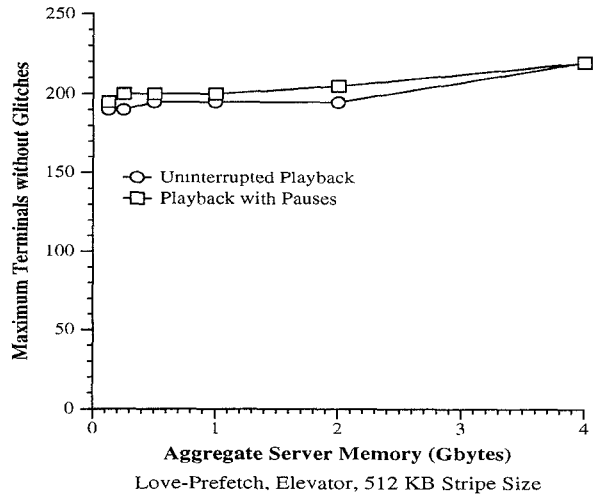


Figure 19: Pausing videos.

Rewind and fast-forward can be similarly supported by simply seeking to a new position within the video, re-priming the terminal's buffers, and starting display from the new position. If a visual search is desired, many alternatives are available. For instance, the terminal can skip forward or backward through the movie showing one or two seconds out of every several seconds of video data. Since the skipped video segments need not be read, this scheme will not significantly increase the load on the video server. Furthermore, the prefetching algorithm at the server can take hints from the terminals so that the appropriate blocks of video are always prefetched next. This scheme does not require any extra resources at the video server or terminals, but does result in a somewhat choppy picture as the terminal skips over portions of the video.

Alternatively, a completely separate version of each movie may be stored for supporting rewind and fast-forward searches [Özde94]. When a user presses rewind or fast-forward, the terminal simply switches from the normal version of the movie to the appropriate position within the appropriate search version. When the user presses play again, the terminal switches back to the normal version again. These switches would require at most a few seconds to re-prime the terminal's buffers. The advantage of this scheme is that for a small amount of additional disk space, the search versions of the movie will provide a smooth, constant rate video stream similar to what a typical VCR produces.

8.2 Piggybacking Terminals

Although the video server does not explicitly try to improve performance by synchronizing two terminals that are watching the same movie (e.g., in a manner similar to [Özde94]), it is possible for two terminals to start the same video at approximately the same time and inadvertently share pages and I/Os. Moreover, there is no reason why the video server could not recognize popular movies and intentionally delay the first subscriber (e.g., by playing a few commercials) while it waits for additional subscribers to request the same movie. In this way, a group of terminals could be "piggybacked" and serviced as though they were one terminal. The piggybacked terminals would place the same demands on the video server as a single terminal would. Since the delay to start a video is not inherently part of the system, its length could be set to any value and could even be selected based on the popularity of the particular movie. Experiments show that a 5 minute delay more than doubles the number of terminals that may be supported glitch-free.

9 Conclusions and Future Work

We have presented the design and performed a detailed simulation analysis of the SPIFFI scalable video-on-demand system. We introduced and analyzed the performance of video server algorithms for real-time disk scheduling, page replacement, and prefetching. We showed that the love prefetch page replacement and delayed prefetching algorithms substantially reduce the memory requirements, and, thus, reduce the cost, of a video server. Furthermore, we demonstrated that while the non-real-time elevator disk scheduling algorithm can function well in a relatively small (16 disk) video server with plenty of memory at the terminals, it does not scale to larger systems. Our real-time disk scheduling algorithm, on the other hand, does scale nearly linearly to at least 64 disks, 256 videos, and 760 terminals.

Our future plans include implementing the SPIFFI scalable video-on-demand system on a cluster of Sun workstations connected by a high-speed network. In addition, we intend to investigate further the algorithms that we outlined for performing rewind and fast forward searches.

References

[Ande91] D. P. Anderson and G. Homsy, "A Continuous Media I/O Server and Its Synchronization Mechanism," *IEEE Computer*, vol. 24, no. 10, pp. 51–57, Oct. 1991.

[Ande92] D. P. Anderson *et al.*, "A File System for Continuous Media," *ACM Trans. on Computing Systems*, vol. 10, no. 4, pp. 311–337, Nov. 1992.

[Bers94] S. Berson *et al.*, "Staggered Striping in Multimedia Information Systems," *Proc. of the 1994 Int. Conf. on the Mgmt. of Data*, pp. 79–90, May 1994.

[Care89] M. J. Carey *et al.*, "Priority in DBMS Resource Scheduling," *Proc. of the 15th Int. Conf. on Very Large Data Bases*, pp. 397–410, Aug. 1989.

[Chan94] E. Chang and A. Zakhor, "Scalable Video Data Placement on Parallel Disk Arrays," *Proc. of Storage and Retrieval for Image and Video Databases II*, pp. 208–221, Feb. 1994.

[Chen93] H.-J. Chen and T.D.C. Little, "Physical Storage Organizations for Time-Dependent Multimedia Data," *Proc. of the 4th Int. Conf. on the Foundations of Data Organizations and Algs.*, pp. 19–34, Oct. 1993.

[Fede94] C. Federighi and L. A. Rowe, "A Distributed Hierarchical Storage Manager for a Video-on-Demand System," *Proc. of Stor. and Retr. for Image and Video Databases II*, pp. 185–197, Feb. 1994.

[Free94] C. S. Freedman *et al.*, "SPIFFI — A Scalable Parallel File System for the Intel Paragon," submitted to *Trans. on Parallel and Distributed Systems*, available on the WWW URL: "http://www.cs.wisc.edu/~freedman/spiffi.ps".

[Gall91] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. of the ACM*, vol. 34, no. 4, pp. 47–58, Apr. 1991.

[Gray93] J. Gray, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, CA, 1993.

[Hari94] J. R. Haritsa and M. B. Karthikeyan, "Disk Scheduling for Multimedia Database Applications," to appear in *COMAD '94*.

[Hask93] R. L. Haskin, "The *Shark* Continuous-Media File Server," *Proc. of the IEEE Computer Conf. Spring '93*, pp. 12–15, Feb. 1993.

[Hsie94] J. Hsieh *et al.*, "Performance of a Mass Storage System for Video-on-Demand," submitted to *Journal of Parallel and Distributed Computing*.

[Laur94] A. Laursen *et al.*, "Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data," *Proc. of the 1994 Int. Conf. on the Mgmt. of Data*, pp. 470–477, May 1994.

[Loug92] P. Lougher and D. Shepherd, "The Design and Implementation of a Continuous Media Storage Server," *Proc. of the 3rd Int. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 69–80, Nov. 1992.

[Özde94] B. Özden *et al.*, "A Low-Cost Storage Server for Movie on Demand Databases," *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pp. 594–605, Sept. 1994.

[Redd94] A. L. N. Reddy and J. C. Wyllie, "I/O Issues in a Multimedia System," *IEEE Computer*, vol. 27, no. 3, pp. 69–74, Mar. 1994.

[Ries78] D. Ries and R. Epstein, "Evaluation of Distribution Criteria for Distributed Database Systems," UCB/ERL Technical Report M78/22, Berkeley, May 1978.

[Rowe92] L. A. Rowe and B. C. Smith, "A Continuous Media Player," *Proc. of the 3rd Int. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 376–386, Nov. 1992.

[Schw90] H. Schwetman, *CSIM Users' Guide*, MCC Technical Report No. ACT-126-90, Microelectronics and Computer Technology Corporation, Austin, TX, Mar. 1990.

[Silb94] A. Silberschatz and P. B. Galvin, *Operating Systems Concepts*, 4th edition, Addison-Wesley, Reading, MA, 1994.

[Suzu94] H. Suzuki *et al.*, "Storage Hierarchy for Video-on-Demand Systems," *Proc. of Storage and Retrieval for Image and Video Databases II*, pp. 198–207, Feb. 1994.

[Teng84] J. Z. Teng and R. A. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance," *IBM Systems Journal*, vol. 23, no. 2, pp. 211–218, 1984.

[Tetz94] W. Tetzlaff *et al.*, "A Methodology for Evaluating Storage Systems in Distributed and Hierarchical Video Servers," *Proc. of the IEEE Computer Conf. Spring '94*, pp. 430–439, Feb. 1994.

[Vin93] H. M. Vin and P. V. Rangan, "Designing a Multiuser HDTV Storage Server," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 1, pp. 152–164, Jan. 1993.

[Yu92] P. S. Yu *et al.*, "Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management," *Proc. of the 3rd Int. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 44–55, Nov. 1992.

[Zipf49] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, 1949.