Disk Striping in Video Server Environments

Banu Özden Rajeev Rastogi Avi Silberschatz

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract

A growing number of applications need access to video data stored in digital form on secondary storage devices (e.g., video-on-demand, multimedia messaging). As a result, video servers that are responsible for the storage and retrieval, at fixed rates, of hundreds of videos from disks are becoming increasingly important. Since video data tends to be voluminous, several disks are usually used in order to store the videos. A challenge is to devise schemes for the storage and retrieval of videos that distribute the workload evenly across disks, reduce the cost of the server and at the same time, provide good response times to client requests for video data.

In this paper, we present schemes that are based on striping videos (fine-grained as well as coarse-grained) across disks in order to effectively utilize disk bandwidth. The proposed schemes are starvation-free and provide good response times to client requests. For the schemes, we show how an optimal-cost server architecture can be determined if data for a certain pre-specified number of videos is to be concurrently retrieved. Finally, through extensive simulations, we compare the performance of the various schemes.

1 Introduction

In recent years, we have witnessed significant advances in both networking technology, and technologies involving the digitization and compression of video. It is now possible to transmit several gigabits of data per second over optic fiber networks. Also, with compression standards like MPEG-1, the bandwidth required for the transmission of a video is as low as 1.5 Mbps. These advances have resulted in a host of new applications involving the transmission of video data over networks. Examples include video on demand, on-line tutorials, training and lectures, multimedia messaging, interactive TV, games, etc.

One of the key components in the above applications is the *video server* which is responsible for the storage and transmission of videos. Depending on the application, a video server may be required to store hundreds of videos, and may be required to concurrently transmit to clients, data for a few hundred to a few thousand videos. Furthermore, the data for every video must be transmitted at a fixed rate depending on the compression technique employed (for MPEG-1, the rate is about 1.5 Mbps).

Due to the voluminous nature of video data (a 100 minute long MPEG-1 video requires approximately 1.125 GB of storage space), and the high cost ($40/MB) of *random access memory* (RAM), storing videos in RAM is prohibitively expensive. A less expensive way of storing videos is to utilize disks, which are much cheaper than RAM (storage on disks costs less than $1/MB). Thus, a video server would be more cost-effective if it relied on disks instead of RAM for the storage of videos. However, modern disks have limited storage capacity (1 GB-9GB) and relatively low transfer rates (30 Mbps - 60 Mbps). As a result, in order to be able to store hundreds of videos as well as support the retrieval of hundreds of videos concurrently, a video server would need to store the videos on several disks. Schemes for laying out the videos on multiple disks are crucial to distributing the load uniformly across the various disks and thus, utilizing the disk bandwidth effectively. For example, certain videos may be more popular than others, and a naive approach that stores every video on an arbitrarily chosen disk could result in certain disks being over-burdened with more requests than they can support, while other disks remain idle.

Another characteristic of disks is that they have a relatively high latency for data access (typically between 10-20 ms). As a result, if very little data is retrieved during each disk access, then the effective transfer rate of the disk is much lower, and thus the number of videos that can be concurrently retrieved is much less. On the other hand, retrieving large amounts of data during each disk access reduces the impact of disk latency and increases the throughput of the disk. However, beyond a certain point, increasing the amount of data retrieved is not cost-effective. The reason for this is that in order to buffer the large amounts of data retrieved, a large amount of RAM is required, and this results in an increase in the cost of the server. What is needed, is an effective method for computing the optimal amount of data to be retrieved during a disk access that reduces the cost of retrieving data per video. Thus, the challenge is to design a low-cost video server that can transmit several videos at the required rate concurrently, utilize the disk bandwidth effectively, and provide low response times to requests for videos.

In this paper, we present starvation-free schemes based on fine-grained as well as coarse-grained striping in order to distribute the workload evenly across disks. For each of the schemes, we compute the optimal amount of data to be retrieved during each disk access such that the number of videos that can be concurrently transmitted is maximized. We show how an optimal cost architecture can be determined if data for a certain pre-specified number of videos is to be retrieved concurrently. For the schemes based on coarse-grained striping, we show how disk bandwidth can be effectively allocated to requests such that there is no starvation. Finally, through extensive simulations, we compare the performance of the various schemes. Our results show that schemes based on coarse-grained striping can support the retrieval of more videos than those based on fine-grained striping. Our simulation results also validate the optimal data retrieval size and optimal cost architecture that we determined analytically.

2 System Model

The *video-server* is a computer system containing one or more processors, RAM, and multiple disks to store videos. Videos are stored on disks in compressed form and need to be displayed at a certain rate, denoted by r_{disp}. The server is connected to clients via a high speed network over videos to clients are transmitted at the rate r_{disp}. We denote by l_i, the length (in bits) of a video V_i.

Clients make requests for videos to the server which maintains a *request list*. Each of the entries in the request list contains an identifier for the requested video (e.g., name) and the set of clients who have requested the video. Requests are added to the request list as follows. If an entry for the requested video is already contained in the request list, then the client is simply added to the set of clients in the entry. If, on the other hand, an entry for the requested video is not contained in the request list, then a new entry containing the video and the client is appended to the end of the request list.

In general, it is not possible to service client requests immediately upon arrival since videos reside predominantly on disk, and they need to be retrieved into RAM before being transmitted to a client. In this paper, we present schemes that the server employs in order to determine the set of videos to be concurrently retrieved from disk at any given time. For each of the videos being retrieved concurrently, the server allocates a certain amount of buffer space in RAM, and retrieves portions of the video into the buffer at certain time intervals such that entire video is retrieved at a rate r_{disp}. The size of the portions retrieved depends on the scheme; thus, the server transmits videos to clients without loading the entire video into RAM. To start a video, a certain initial amount of the video data must be retrieved into RAM. Once this is accomplished, the transmission of the video to all the clients with outstanding requests for the video can commence. We refer to the continuous transfer of a video from disk to RAM at rate r_{disp}, as a *stream*.

inner track transfer rate	r_{disk}	45 Mbps
Settle time	t_{settle}	0.6 ms
Seek time (worst case)	t_{seek}	17 ms
Rotational latency (worst case)	t_{rot}	8.34 ms
Worst case latency	t_{lat}	25.5 ms
Cost	C_d	\$1500
Capacity		2 GB

Figure 1: Disk Parameters for a Commercially Available Disk

For the development of the schemes, it is important to understand the characteristics of disks, which we now describe. Data on disks is stored in a series of concentric circles, or *tracks*, and accessed using a disk head. Disks rotate on a central spindle and the speed of rotation determines the transfer rate of disks. Data on a particular track is accessed by positioning the head on (also referred to as *seeking* to) the track containing the data, and then waiting until the disk rotates enough so that the head is positioned directly above the data. Seeks typically consist of a coast during which the head moves at a constant speed and a settle, when the head position is adjusted to the desired track. Thus, the latency for accessing data on disk is the sum of seek and rotational latency. In the table of Figure 1, we present the notation and values employed in the paper for the various disk characteristics (the values are for a commercially available disk [1]). In this paper, we assume that $r_{disp} < r_{disk}$, for the videos.

As mentioned earlier, a video server that stores hundreds of videos would require a large number of disks to hold the video data. A naive storage scheme in which an entire video is stored on a single disk could result in an ineffective utilization of disk bandwidth. The reason for this is that not all videos are accessed with the same frequency; clients may request to view certain videos more often than others, thus causing certain disks to be busy while others, with less frequently requested videos, remain idle. Also, unless a video is replicated on several disks, the number of concurrent streams of the video that can be supported is bounded above by the bandwidth of the disk storing the video.

In this paper, we store videos using *disk striping*, a popular technique in which consecutive logical data units (referred to as *stripe units*) are distributed among the disks in a round-robin fashion [9, 4, 2]. Disk striping, in addition to distributing the workload uniformly across disks, also enables multiple concurrent streams of a video to be supported without having to replicate the video.

3 Related Work

A number of schemes for retrieving video data continuously from disks have been proposed in the literature [12, 6, 7, 8, 10, 3, 2, ?]. Most of them, however, only consider the problem of retrieving videos from a single disk, and do not address the issue of evenly distributing the workload in the case that multiple disks are employed to store the videos.

Among striping-based schemes, *staggered striping* was proposed in [2] to utilize disk bandwidth effectively for both high bandwidth (that is, those for which $r_{disp} \geq r_{disk}$) as well as low bandwidth (those for which $r_{disp} < r_{disk}$) videos with different rate requirements. For the high bandwidth videos, each video object is divided into subobjects and each subobject is further declustered across $\lceil \frac{r_{disp}}{r_{disk}} \rceil$ disks, from which it is retrieved in parallel. The data for a subobject on a disk is referred to as a *fragment*, and the distance between disks containing the first fragment for consecutive subobjects is referred to as the *stride*. The authors show that by choosing the stride to be relatively prime with respect to the number of disks, the workload can be uniformly distributed across the disks. The authors also show that a fragment size of about 2 cylinders wastes only 10% of the bandwidth, and at the same time provides a reasonable display latency time.

The striping schemes and results presented in [5, 4, 9] are more general, and are not specifically tailored for video server environments in which workloads typically comprise of accesses (to large files) that are sequential in nature, and have timing requirements.

4 Fine-grained Striping

In fine-grained striping, the striping unit is typically a bit, a byte or a sector [4, 9]. As a consequence, if there are m disks, then every retrieval involves all the m disk heads, and the m disks behave like a single disk with bandwidth $m \cdot r_{disk}$. This striping technique is followed in the RAID-3 data distribution scheme [9].

4.1 Servicing Requests

We begin by describing the scheme followed by the server in order to retrieve data for multiple concurrent video streams. The server maintains a *service list* that is different from the request list , and contains the videos for which data is being retrieved. The server retrieves data for the videos in the service list in *rounds*, the number of bits retrieved for a video during a round being d. The d bits retrieved for a video during a round follow the d bits retrieved for the video during the previous round. The value of d is computed based on the amount of RAM, D, and the number of disks, m, such that the number of concurrent streams that can be supported is maximized. In addition, it is a multiple of $m \cdot su$, where su is the stripe unit size (we describe the method used for computing d at the end of this subsection). A video can thus be viewed as a sequence of portions of size d; each portion can further be viewed as a sequence of sub-portions, each of size $m \cdot su$. Each sub-portion is striped across the m disks; the stripe units contained in a sub-portion are at the same position on their respective disks. In addition, stripe units on a disk, belonging to consecutive sub-portions of a portion, are stored contiguously on the disk one after another.

At the start of a round, the server sorts the videos in the service list based on the positions of the tracks on disk, of the d bits to be retrieved for each video. It then retrieves d bits from disk for each of the videos in the order of their appearance in the service list. Note that sorting the videos ensures that the disk heads move in a single direction when servicing videos during a round. As a result, random seeks to arbitrary locations are eliminated (this is similar to the known C-SCAN disk scheduling algorithm [11]). Since the time to transmit d bits of video is $\frac{d}{r_{disp}}$, in order to ensure that data for every video is retrieved at rate r_{disp}, we require every round to begin $\frac{d}{r_{disp}}$ after the previous round. As a result, we require the time to service videos during a round to never exceed $\frac{d}{r_{disp}}$, the duration of a round. Since, during a round, disk heads travel across the disk at most twice, and retrieving data for each video, in the worst case, incurs a settle and a worst-case rotational latency overhead, we require the following equation to hold (assuming that $V_1, V_2, \ldots, V_q$ are the videos in the service list):

$$2 \cdot t_{seek} + q \cdot \left(\frac{d}{m \cdot r_{disk}} + t_{rot} + t_{settle}\right) \leq \frac{d}{r_{disp}} \quad (1)$$

At the start of a round, before the server sorts the videos in the service list, it adds videos at the head of the request list to the service list as long as Equation 1 holds for videos in the service list after each of the videos is added (videos that are added to the service list are deleted from the head of the request list). For every video added to the service list, a buffer of size $2 \cdot d$ is allocated. The d bits for the video during successive rounds are retrieved into the buffer. Transmission of a video's bits to clients is begun at the end of the round in which the first d bits of the video have been retrieved into it's buffer. The reason for this is that since new videos may be added to the service list and videos are sorted at the start of each round, the times (relative to the start of a round) at which the d bits for a video are retrieved, in two consecutive rounds, may not be the same. Thus, by ensuring that a video's buffer contains at least d bits at the start of each round, we ensure that bits for the video can be transmitted at a rate r_{disp} irrespective of when during the round the next d bits are retrieved for the video. Finally, a video is deleted from the service list at the end of the round in which all the data for it has been retrieved. The above approach to retrieving data for videos is similar to the *streaming* RAID scheme proposed in [12].

We now address the issue of computing a value for d based on the amount of RAM, D, and the number of disks, m, such that the number of concurrent streams that can be supported is maximized. From Equation 1, it follows that, for a given value of d, the maximum number of streams that can be supported is $\frac{\frac{d}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d}{m \cdot r_{disk}} + t_{rot} + t_{settle}}$. Thus, since $r_{disp} < m \cdot r_{disk}$, as the value of d increases, the number of streams that can be supported also increases. However, since the buffer per stream is $2 \cdot d$ and the total buffer requirements must not exceed D, increasing d beyond a certain value results in a decrease in the number of streams. Thus, the value of d that supports the maximum number of streams, can be obtained by solving the following equation:

$$\frac{2 \cdot d \cdot (\frac{d}{r_{disp}} - 2 \cdot t_{seek})}{\frac{d}{m \cdot r_{disk}} + t_{rot} + t_{settle}} = D$$

Let $d_{calc} = \frac{r_{disp}}{4} \cdot (4 \cdot t_{seek} + \frac{D}{m \cdot r_{disk}} + \sqrt{(4 \cdot t_{seek} + \frac{D}{m \cdot r_{disk}})^2 + \frac{8 \cdot D \cdot (t_{rot} + t_{settle})}{r_{disp}}})$ be the maximum value for d obtained as a result of solving the above equation. Let $q_{calc} = \frac{\frac{d_{calc}}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d_{calc}}{m \cdot r_{disk}} + t_{rot} + t_{settle}}$ be the maximum number of streams that can be supported with $d = d_{calc}$. Note that $2 \cdot d_{calc} \cdot q_{calc} = D$. The problem is that d_{calc} may not be a multiple of $m \cdot su$. For all values of d greater than d_{calc}, the maximum number of streams is $\frac{D}{2 \cdot d}$; on the other hand, for values of d less than d_{calc}, the maximum number of streams is $\frac{\frac{d}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d}{m \cdot r_{disk}} + t_{rot} + t_{settle}}$. Thus, the optimal value for d is either $\lceil \frac{d_{calc}}{m \cdot su} \rceil \cdot m \cdot su$ or $\lfloor \frac{d_{calc}}{m \cdot su} \rfloor \cdot m \cdot su$ depending on which of the following two is greater: (1) $\frac{D}{2 \cdot \lceil \frac{d_{calc}}{m \cdot su} \rceil \cdot m \cdot su}$ or (2) $\frac{\frac{d}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d}{m \cdot r_{disk}} + t_{rot} + t_{settle}}$ with $d = \lfloor \frac{d_{calc}}{m \cdot su} \rfloor \cdot m \cdot su$.

Thus, if $D = 2$ Gb and $m = 50$, then the optimal value for d is 4.0 Mb, and the maximum number of MPEG-1 streams that can be supported is 248 (the values for disk parameters are those presented in Figure 1 and su is chosen to be 512 bytes). For simplicity, in the schemes presented in this section and later in the paper, video streams are begun only based on the availability of disk bandwidth. As a result, we select d to be $\lfloor \frac{d_{calc}}{m \cdot su} \rfloor \cdot m \cdot su$. The schemes can be extended to take into account even the availability of buffer space, in which case, an optimal value for d can be selected as described above. However, we do not discuss these extensions in this paper.

4.2 Designing Video Servers

Ideally, the architecture of a video server must be such that it can support a certain pre-specified number of concurrent streams and its cost is as low as possible. In the following, we compute the amount of RAM, D, and the number of disks, m, that a video server must contain if it is to support Q concurrent streams, and its cost is to be optimal.

Let C_r denote the cost of RAM per Mb and C_d be the cost of a single disk. For a given value of m, the minimum amount of RAM required to support Q streams is $2 \cdot Q \cdot d$, where $d = \lceil \frac{\hat{d}}{m \cdot su} \rceil \cdot m \cdot su$ and $\hat{d} = \frac{(Q \cdot (t_{rot} + t_{settle}) + 2 \cdot t_{seek}) \cdot m \cdot r_{disk} \cdot r_{disp}}{(m \cdot r_{disk} - Q \cdot r_{disp})}$ is the minimum value of d required to support Q streams, obtained as a result of solving Equation 1. Thus, $D = 2 \cdot Q \cdot d$, and $C(m)$, the minimum cost of the server for a given m, is

$$C(m) = C_r \cdot 2 \cdot Q \cdot d + C_d \cdot m$$

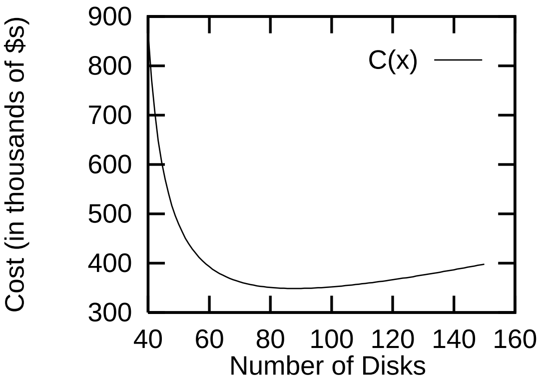

Figure 2: Cost v/s number of Disks

Substituting for d in the above equation, we obtain the following

$$C(m) = C_r \cdot 2 \cdot Q \cdot \lceil \frac{(Q \cdot (t_{rot} + t_{settle}) + 2 \cdot t_{seek}) \cdot r_{disk} \cdot r_{disp}}{su \cdot (m \cdot r_{disk} - Q \cdot r_{disp})} \rceil \cdot m \cdot su + C_d \cdot m$$

From the above cost equation, it follows that it may not be cost-effective to architect the server with the minimum number of disks required to support Q streams (that is, m is such that $m \cdot r_{disk} > Q \cdot r_{disp} \geq (m-1) \cdot r_{disk}$). The reason for this is that the size of the buffer per stream, $2 \cdot d$, is inversely proportional to $r_{disk} - \frac{Q \cdot r_{disp}}{m}$. Thus, at $m = \frac{Q \cdot r_{disp}}{r_{disk}}$, the buffer size is infinity, and subsequent increases in the number of disks, m, cause the size of the buffer per stream to decrease; the decrease in the buffer size is larger for smaller values of m. Thus, since RAM is more expensive than disks, the reduction in the cost of the buffer for the streams due to increasing the number of disks beyond $\frac{Q \cdot r_{disp}}{r_{disk}}$, could more than offset the cost of the additional disks. In Figure 2, we show how the cost of the server, $C(m)$, that supports 1000 MPEG-1 streams varies with the number of disks, m (the values for disk parameters are those presented in Figure 1, C_r is \$5/Mb, and su is 512 bytes). As the number of disks increases, initially the cost decreases due to the large decreases in buffer sizes. However, beyond a certain number of disks, the cost of a disk exceeds the decrease in the cost of RAM as a result of the decrease in buffer sizes, and thus the cost of the server increases.

Our goal is to compute the value of m for which $C(m)$ is minimum, subject to the constraint that $m > \frac{Q \cdot r_{disp}}{r_{disk}}$. We begin by computing the value of m for which $C(m)$ without the $\lceil\ \rceil$s, which we denote by $\hat{C}(m)$, is minimum. Solving $\frac{d}{dm}(\hat{C}(m)) = 0$, we obtain the following value for m, which we denote by m_{calc}, that satisfies the constraint and at which $\hat{C}(m)$ is minimum.

$m_{calc} = \frac{1}{r_{disk}}(Q \cdot r_{disp} +$

$\sqrt{\frac{2\cdot Q^2\cdot C_r\cdot r_{disp}^2\cdot r_{disk}\cdot(Q\cdot(t_{rot}+t_{settle})+2\cdot t_{seek})}{C_d}})$

Since $\frac{d}{dm}(\frac{d}{dm}(\hat{C}(m))) > 0$ at $m = m_{calc}$, it follows that for increasing values of m beyond m_{calc} and decreasing values of m below m_{calc} (and above $\frac{Q\cdot r_{disp}}{r_{disk}}$), the value of $\hat{C}(m)$ increases. The optimal value of m for which $C(m)$ is minimal can now be computed as follows.

We refer to values of m for which $\hat{C}(m) = C(m)$ as breakpoints. At increasing breakpoints beyond m_{calc}, the value of $C(m)$ increases. Furthermore, in between two successive breakpoints beyond m_{calc}, the value of $C(m)$ increases linearly with m. As a result, the optimal value of m is bounded above by $\lceil m_{calc}\rceil$. Furthermore, since for a given value of m, $C(m)$ is always greater than or equal to $\hat{C}(m)$, the lower bound for the optimal value of m can be obtained as follows. Let $C_{\lceil calc\rceil}$ be the value of $C(m)$ at $m = \lceil m_{calc}\rceil$. By solving $\hat{C}(m) = C_{\lceil calc\rceil}$, two values of m for which $\hat{C}(m) = C_{\lceil calc\rceil}$ are obtained. One of these is to the right of m_{calc}, and the other is to the left of m_{calc}, which we denote by m_{left}. Since $\hat{C}(m)$ increases to the left of m_{left} and $C(m)$ is at least as high as $\hat{C}(m)$, the optimal value of m is an integer in the interval between m_{left} and $\lceil m_{calc}\rceil$ for which $C(m)$ is minimum.

The optimal value of m for a server that supports 1000 MPEG-1 streams is 88. The amount of RAM required is 43.3 Gb and the cost of the the server is \$348,689. Also, the value for d is 21.6 Mb, and the duration of a round is 14.4 s.

5 Coarse-grained Striping

In coarse-grained striping, the size of stripe units is much larger; it is the amount of data typically retrieved during a single disk access [2, 4, 9]. Thus, in contrast to fine-grained striping in which all disk arms are involved in data retrieval during each access, in coarse-grained striping, usually, only a single disk is involved. For large requests and for sequential accesses to data, coarse-grained striping distributes the workload evenly among the various disks. This striping technique is followed in the RAID-5 data distribution scheme [9].

5.1 Servicing Requests

In the coarse-grained striping scheme, the size of a stripe unit is d, the number of bits retrieved for a video during each round (we show how d is computed at the end of this subsection). Every video is assumed to have a length that is a multiple of d[1]. The various videos are first concatenated to form a *super-video*. Consecutive stripe units of size d belonging to the *super-video* are then stored on the m disks in a round-robin fashion. We denote the disk on which the first stripe unit of a video V_i is stored by $disk(V_i)$. Note that, with the coarse-grained striping scheme, for two different videos V_i and V_j, $disk(V_i)$ may be different from $disk(V_j)$. We now describe schemes for servicing client requests given that the videos are laid out using coarse-grained striping.

In contrast to the fine-grained striping scheme, in the coarse-grained striping scheme, a service list is maintained for every disk. The service list for a disk contains the videos for which data is being retrieved from the disk during the current round. The data for a video is retrieved into a buffer of size $2\cdot d$ for the video. As before, at the beginning of a round, the videos in each service list are sorted based on the positions of the tracks on disk, of the d bits to be retrieved for each of the videos during the round. Following this, the d bits for every video in the service list for a disk are retrieved from the disk in the sorted order. Furthermore, data from different disks is retrieved in parallel. As a result, in order to retrieve data at a rate r_{disp}, since rounds must begin at intervals of $\frac{d}{r_{disp}}$, for every service list (let $V_1, \ldots, V_q$ be the videos in the service list), we require the following equation to hold:

$$q\cdot(\frac{d}{r_{disk}} + t_{rot} + t_{settle}) + 2\cdot t_{seek} \leq \frac{d}{r_{disp}} \qquad (2)$$

At the end of a round, videos for which no more data needs to be retrieved are deleted from the service lists. Following this, the service list for every disk is set to the service list of the disk preceding it; the data to be retrieved for a video in the service list being the d bits in the video that follow the d bits retrieved for the video in the previous round from the preceding disk. Thus, data for a video during successive rounds is retrieved from successive disks. Also, at the end of the round in which the first d bits have been retrieved for a video, transmission of data to clients that requested it is begun. The above approach to storing and retrieving data is similar to the staggered striping scheme for low bandwidth objects, proposed in [2].

We now present schemes that address the issue of when to begin retrieving data for videos in the request list or alternately, when to insert videos contained in the request list into the service list for a disk. All of the schemes are employed by the server just before the service lists are sorted at the beginning of a round, and after the service list for every disk is set to that of its preceding disk, at the end of the previous round. Note that if q is the number of videos serviced at a disk during a round, then the *unutilized* (also referred to as *wasted* or *available*) bandwidth on the disk, during the round, is at least $\frac{d}{r_{disp}} - q\cdot(\frac{d}{r_{disk}} + t_{rot} + t_{settle}) - 2\cdot t_{seek}$.

The simplest scheme is to perform the following action for every request in the request list, beginning with the first request.

> If for the requested video V_i and the videos contained in the service list for $disk(V_i)$, Equation 2 holds, then the request is deleted from the request list and added to the service list for $disk(V_i)$.

The problem with the scheme is that, even though it utilizes disk bandwidth fairly well, certain requests in

[1] This can be achieved by appending advertisements or padding videos at the end.

the request list may never be serviced; thus causing them to wait forever.

Example 1: Consider a server with 100 disks numbered 0 through 99. Consider videos $V_1, \ldots, V_{998}$, each of which begins at disk 0 and terminates at disk 99, video V_{999} that begins on disk 0 and terminates at disk 49, and video V_{1000} that begins at disk 50. Now, suppose a request for video V_{1000} arrives at a time when the bandwidth of all the disks is being used to retrieve data for videos in $\{V_1, \ldots, V_{998}\}$. Furthermore, suppose that additional requests for videos in $\{V_1, \ldots, V_{998}\}$ keep arriving at the server. Every time data retrieval for a video in $\{V_1, \ldots, V_{998}\}$ completes, bandwidth is available on disk 0. Thus, since $disk(V_{1000}) = 50$ while videos in $\{V_1, \ldots, V_{998}\}$ begin at disk 0, the above scheme would allocate the available bandwidth to one of the videos in $\{V_1, \ldots, V_{998}\}$, for which a request is contained in the request list. □

A variant of the above scheme that adds a video to a service list only after videos preceding it in the request list have been added to service lists, eliminates the above problem. However, a problem with the approach is that it could result in disk bandwidth being unnecessarily wasted. In Example 1, presented earlier, suppose that there are outstanding requests for videos V_1 and V_{1000} (the one for V_1 preceding the one for V_{1000} in the request list), and bandwidth is available only on disks 50 and 98. With the modified scheme, since V_1 can be added to the service list of disk 0 only when bandwidth becomes available on disk 0 (after two rounds), V_{1000} cannot be added to the service list for disk 50 even though bandwidth is available on it. Thus, the bandwidth on disk 50 is wasted.

In the following, we present the *current available bandwidth* (CAB) scheme that prevents requests from starving and at the same time, utilizes disk bandwidth fairly effectively. In order to prevent starvation, the CAB scheme reserves available disk bandwidth for videos based on the order in which they appear in the request list. The assignment of reserved disk bandwidth to videos is such that the amount of disk bandwidth that is wasted is minimized. The CAB scheme requires two additional data structures to be maintained. One of the data structures is a list of (disk,video) ordered pairs which stores information about the disk on which bandwidth for a video is reserved. We refer to this list as the *reserved list*. The other data structure utilized by our scheme is a *free list*, which stores information relating to the available bandwidth on each disk; the free list entries are (disk, available bandwidth) ordered pairs. For each reserved and free list entry e, $e(1)$ and $e(2)$ denote the first and second elements of e, respectively. We assume the m disks are numbered 0 through $m-1$. The distance from disk i to disk j is denoted by $dist(i,j)$. If $i \leq j$, then $dist(i,j)$ is simply $j-i$; else $dist(i,j)$ is $m-i+j$. Note that $dist(i,j)$ may not be equal to $dist(j,i)$. The distance between the disk on which available bandwidth is reserved for a video V_i and $disk(V_i)$ is the number of rounds for which $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$ portion of a disk's bandwidth would be unutilized due to video V_i. It is also the number of rounds after which data retrieval for V_i is begun. The available bandwidth for each disk in the free list is $\frac{d}{r_{disp}} - 2 \cdot t_{seek}$ initially. The reserved list is initially empty.

The steps of the CAB scheme, for inserting videos from the request list into service lists at the start of a round just before service lists are sorted, are enumerated below.

1. For every reserved list and free list entry e, $e(1)$ is set to $(e(1)+1) \bmod m$; (thus, the disk in each entry is set to the disk following it).

2. For every video for which the last d bits were retrieved in the previous round from disk i, we add the bandwidth available on disk $(i+1) \bmod m$ to the free list as follows. Two variables, cur_entry and cur_disk are maintained; cur_entry is initially set to the first entry in the reserved list and cur_disk is set to $(i+1) \bmod m$.

   (a) If cur_entry is nil, then add $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$ to the available bandwidth of the free list entry for cur_disk, and go to Step 3.

   (b) If $dist(\text{cur\_disk}, disk(\text{cur\_entry}(2)))$ is less than $dist(\text{cur\_entry}(1), disk(\text{cur\_entry}(2)))$, then the values of cur_entry(1) and cur_disk are swapped.

   (c) cur_entry is set to the next entry in the reserved list. Go to Step 2(a).

3. Once all the available disk bandwidth has been added to the free list, it is then reserved for requests in the request list as follows.

   (a) If the request list is empty or for every disk, the available bandwidth in the free list entry for the disk is less than $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$, go to Step 4. Else, set cur_video to the video contained in the first request in the request list, and delete the request from the request list.

   (b) Let disk j be the disk for which available bandwidth in the free list is at least $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$, and $dist(j, disk(\text{cur\_video}))$ is minimum. Append $(j, \text{cur\_video})$ to the end of the reserved list.

   (c) Decrement $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$ from the available bandwidth for disk j in the free list. Go to Step 3(a).

4. For all entries e in the reserved list such that $e(1) = disk(e(2))$, delete e from the reserved list and insert $e(2)$ into the service list for $e(1)$ (the data to be retrieved for $e(2)$ is the first d bits).

In Step 1, the bandwidth available on a disk is set to the bandwidth available on the disk preceding it. It may be possible that for newly available bandwidth on disk $(i+1) \bmod m$ (due to the completion of data retrieval for a video on disk i), swapping it with the disk assigned to some video in the reserved list could result in better bandwidth utilization. Thus, in Step 2, for a video V_j in the reserved list, if disk $(i+1) \bmod m$ is closer to $disk(V_j)$ than the disk currently assigned to V_j, then disk $(i+1) \bmod m$ is assigned to V_j. The available bandwidth on the disk that was previously assigned to V_j is then similarly assigned to some subsequent video in the reserved list; the assignments are repeated until the end of the reserved list is reached. In Step 3, available bandwidth in the free list is assigned to requests at the head of the request list, and finally, in Step 4, videos in the reserved list for which data retrieval can begin, are added to the service lists. The following example illustrates the actions performed by the various steps of the scheme.

Example 2: Consider the server in Example 1 with 100 disks and videos $V_1, \ldots, V_{1000}$. Suppose, at the start of a round, say r_0, the request list contains requests for videos V_1 and V_{1000} (the one for V_1 preceding the one for V_{1000}), the reserved list is empty, and the only disks with available bandwidth greater than or equal to $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$ in the free list are disks 30 and 48. Due to Step 3, since $dist(48, disk(V_1)) < dist(30, disk(V_1))$ ($disk(V_1)$ is 0), $(48, V_1)$ is first appended to the reserved list, followed by $(30, V_{1000})$. Suppose, during round r_0, data retrieval for video V_2 completes at disk 99. At the start of round r_0+1, the available bandwidth on disk 0 is added to the free list as follows. During the first iteration of Step 2, since $dist(0, disk(V_1)) < dist(49, disk(v_1))$, $\mathsf{cur\_disk}$ is set to 49 and the first reserved list entry becomes $(0, disk(V_1))$. During the second iteration, since $dist(49, disk(V_{1000})) < dist(31, disk(V_{1000}))$, $\mathsf{cur\_disk}$ is set to 31 and the second reserved list entry becomes $(49, V_{1000})$. At the end of Step 2, $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$ is added to the available bandwidth of the free list entry for disk 31. Due to Step 4, since $disk(V_1) = 0$, V_1 is inserted into the service list for disk 0 at the start of round r_0+1; furthermore, since $disk(V_{1000}) = 50$, V_{1000} is inserted into the service list for disk 50 at the start of round r_0+2. □

Note that, in Step 2, since a video V_i in the reserved list is not re-assigned to a different disk unless the disk is closer to $disk(V_i)$, data retrieval for no video in the request list is delayed forever; thus, the scheme is starvation-free. Also, the CAB scheme incurs very little overhead. The availability of new disk bandwidth requires a comparison with every entry in the reserved list. Furthermore, the addition of a new request into the reserved list requires a comparison with every other disk entry in the free list (in order to determine the closest disk with the required available bandwidth).

We next show that the assignment of available bandwidth to videos in the reserved list due to the CAB scheme results in as good or better disk utilization than any other assignment of the available disk bandwidth to the same videos. Note that, for an entry e in the reserved list, $dist(e(1), disk(e(2)))$ is the number of rounds that bandwidth for a stream is wasted. Thus, we show that the scheme minimizes the wasted bandwidth by showing that for all entries e in the reserved list, the sum total of $dist(e(1), disk(e(2)))$ is less than or equal to the total distance due to any other assignment of videos in the reserved list to disks with available bandwidth in the reserved and free lists. It can be shown that the following invariant is preserved by the steps of the CAB scheme.

> For every entry e in the *reserved* list, for all disks k contained in (1) entries following e in the reserved list, and (2) entries in the free list such that the available bandwidth is at least $\frac{d}{r_{disk}} + t_{rot} + t_{settle}$, $dist(e(1), disk(e(2))) \leq dist(k, disk(e(2)))$.

The above invariant is trivially preserved when an entry e is appended to the end of the reserved list in Step 3, since among disks with available bandwidth in the free list, $dist((e_1), disk(e(2)))$ is minimum. When bandwidth on a disk becomes available, every iteration in Step 2 of the scheme also preserves the invariant, and in addition, the following property: for entries e preceding $\mathsf{cur\_entry}$, $dist(e(1), disk(e(2))) \leq dist(\mathsf{cur\_disk}, disk(e(2)))$. We show that Step 2(b) preserves both the invariant and the property. The invariant and property is preserved for all entries e preceding $\mathsf{cur\_entry}$ as a result of Step 2(b), since just before Step 2(b), $dist(e(1), disk(e(2))) \leq dist(\mathsf{cur\_disk}, disk(e(2)))$ and $dist(e(1), disk(e(2))) \leq dist(\mathsf{cur\_entry}(1), disk(e(2)))$, and the only action that Step 2(b) can perform is swap $\mathsf{cur\_entry}(1)$ and $\mathsf{cur\_disk}$. Furthermore, since due to Step 2(b), $dist(\mathsf{cur\_entry}(1), disk(\mathsf{cur\_entry}(2)))$ can only decrease, and $dist(\mathsf{cur\_entry}(1), disk(\mathsf{cur\_entry}(2))) \leq dist(\mathsf{cur\_disk}, disk(\mathsf{cur\_entry}(2)))$ at the end of Step 2(b), the invariant and property are preserved for $\mathsf{cur\_entry}$.

We now show that for any two entries (i, V_j) and (k, V_l) in the reserved list ((i, V_j) preceding (k, V_l)), assigning disk i to V_l and disk k to V_j does not cause the total distance to decrease. The reason for this is that, due to the invariant, $dist(i, disk(V_j)) \leq dist(k, disk(V_j))$, and since distances between disks are linear, it follows that $dist(i, disk(V_j)) + dist(k, disk(V_l)) \leq dist(i, disk(V_l)) + dist(k, disk(V_j))$. The following theorem, thus, follows from the above invariant.

Theorem 1: Consider an arbitrary assignment A of videos in the reserved list to disks with available bandwidth in the reserved and free lists ($A(V_i)$ is the disk assigned to video V_i). The sum of $dist(A(V_i), disk(V_i))$ for all videos assigned in A is greater than or equal to the sum of $dist(e(1), disk(e(2)))$ for all entries e in the reserved list. □

5.2 Determining an Optimal Stripe Unit Size

We now address the issue of determining an optimal value for d such that the number of concurrent streams that can be supported by the server is maximized. For a given value of d, the maximum number of streams that can be supported by each disk is $\frac{\frac{d}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d}{r_{disk}} + t_{rot} + t_{settle}}$, the maximum value for q obtained as a result of solving Equation 2 (and by the server, it is m times the amount). Furthermore, since the buffer per stream is $2 \cdot d$, and the total buffer requirements cannot exceed D, the optimal value for d can be obtained by solving the following equation:

$$2 \cdot m \cdot d \cdot \left(\frac{\frac{d}{r_{disp}} - 2 \cdot t_{seek}}{\frac{d}{r_{disk}} + t_{rot} + t_{settle}}\right) = D$$

Let $d_{calc} = \frac{r_{disp}}{4} \cdot (4 \cdot t_{seek} + \frac{D}{m \cdot r_{disk}} + \sqrt{(4 \cdot t_{seek} + \frac{D}{m \cdot r_{disk}})^2 + \frac{8 \cdot D \cdot (t_{rot} + t_{settle})}{m \cdot r_{disp}}})$ be the maximum value of d obtained as a result of solving the above equation. Let q_{calc} be the maximum number of streams that can be supported by each disk for $d = d_{calc}$. Note that $2 \cdot d_{calc} \cdot q_{calc} \cdot m = D$. The problem is that q_{calc} may not be an integer. As a result, the maximum number of concurrent streams that can be supported by the server is $\lfloor q_{calc} \rfloor \cdot m$. Thus, since the RAM required to support $\lfloor q_{calc} \rfloor \cdot m$ streams may be substantially less than D, if $d_{\lceil calc \rceil}$ is the minimum value of d in order to support $\lceil q_{calc} \rceil$ streams from a disk, then by choosing the value of d to be $d_{\lceil calc \rceil}$, it may be possible for the server to support a larger number of concurrent streams. Since $d_{\lceil calc \rceil} \geq d_{calc}$, if $d = d_{\lceil calc \rceil}$, then the maximum number of streams that can be supported by the server is $\lfloor \frac{D}{2 \cdot d_{\lceil calc \rceil}} \rfloor$. Depending on which of $\lfloor q_{calc} \rfloor \cdot m$ or $\lfloor \frac{D}{2 \cdot d_{\lceil calc \rceil}} \rfloor$ is greater, the value of d is set to either d_{calc} or $d_{\lceil calc \rceil}$.

Thus, if $D = 2$ Gb and $m = 50$, then the optimal value of d is .9 Mb, and the maximum number of streams that can be supported is 1011. This is more than 4 times the number of streams that can be supported by the fine-grained striping scheme for the same values of D and m (see Section 4.1). The reason for this is that the latency overhead per data access for the fine-grained scheme is m times the overhead per data access for the coarse-grained scheme. As a result, the value of d_{calc} computed for the fine-grained scheme (in Section 4.1) is larger than that computed for the coarse-grained scheme.

For simplicity, in the schemes presented in this section and later in the paper, video streams are begun only based on the availability of disk bandwidth. As a result, we select d to be d_{calc}. The schemes can be extended to take into account even the availability of buffer space, in which case, an optimal value for d can be selected as described above. However, we do not discuss these extensions in this paper.

5.3 Designing Video Servers

We next address the issue of architecting an optimal-cost video server that supports a certain prespecified number, Q, of concurrent streams. If the server has m disks, then in order to support Q streams

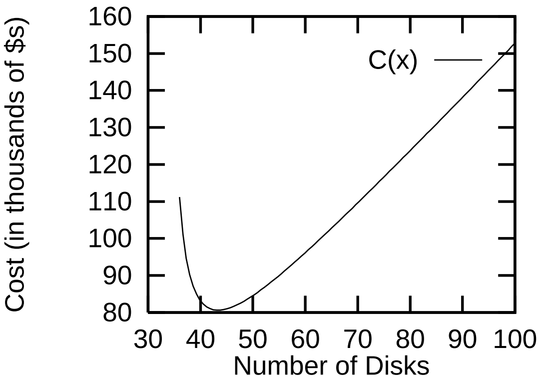

Figure 3: Cost v/s number of disks

from the m disks, d must be such that data for $\lceil \frac{Q}{m} \rceil$ concurrent streams can be retrieved from each of the disks. Thus, from Equation 2, it follows that $d = \frac{(\lceil \frac{Q}{m} \rceil \cdot (t_{rot} + t_{settle}) + 2 \cdot t_{seek}) \cdot r_{disk} \cdot r_{disp}}{r_{disk} - \lceil \frac{Q}{m} \rceil \cdot r_{disp}}$, and the minimum amount of RAM required D is $2 \cdot Q \cdot d$. As a result, the cost of the server, in terms of the number of disks, obtained as a result of substituting for d in $C_r \cdot 2 \cdot Q \cdot d + C_d \cdot m$ is as follows.

$$C(m) = C_r \cdot 2 \cdot Q \cdot r_{disk} \cdot r_{disp} \cdot \frac{(\lceil \frac{Q}{m} \rceil \cdot (t_{rot} + t_{settle}) + 2 \cdot t_{seek})}{r_{disk} - \lceil \frac{Q}{m} \rceil \cdot r_{disp}} + C_d \cdot m$$

In Figure 3, we show how $C(m)$ varies with the number of disks, m, for a server that is required to support 1000 MPEG-1 streams. We are interested in computing the value of m for which $C(m)$ is minimum subject to the constraint that $m > \frac{Q \cdot r_{disp}}{r_{disk}}$. Let $\hat{C}(m)$ denote $C(m)$ without the $\lceil \; \rceil$s. The value of m for which $\hat{C}(m)$ is minimum, denoted by m_{calc}, can be obtained by solving $\frac{d}{dm}\hat{C}(m) = 0$, and is as follows.

$$m_{calc} = \frac{1}{r_{disk}}(Q \cdot r_{disp} + \sqrt{\frac{2 \cdot Q^2 \cdot C_r \cdot r_{disk} \cdot r_{disp} \cdot (2 \cdot t_{seek} \cdot r_{disp} + r_{disk} \cdot (t_{rot} + t_{settle}))}{C_d}})$$

Also, since $\frac{d}{dm}(\frac{d}{dm}(\hat{C}(m)))$ is positive at $m = m_{calc}$, for increasing values of m greater than m_{calc}, and for decreasing values of m below m_{calc} and above $\frac{Q \cdot r_{disp}}{r_{disk}}$, the value of $\hat{C}(m)$ increases. We now compute the value of m for which $C(m)$ is minimum. We refer to values of m for which $C(m) = \hat{C}(m)$ as breakpoints. The value of $C(m)$ increases at successive breakpoints beyond m_{calc}. Furthermore, between successive breakpoints, the value of $C(m)$ increases with m. Thus, the optimal value of m cannot exceed $\lceil m_{calc} \rceil$. Let $C_{\lceil calc \rceil}$ be the value of $C(m)$ at $\lceil m_{calc} \rceil$. Solving the equation $\hat{C}(m) = C_{\lceil calc \rceil}$ yields two roots, one of which is to the right of m_{calc} and the other to the left of m_{calc}, which we denote by m_{left}. Since for

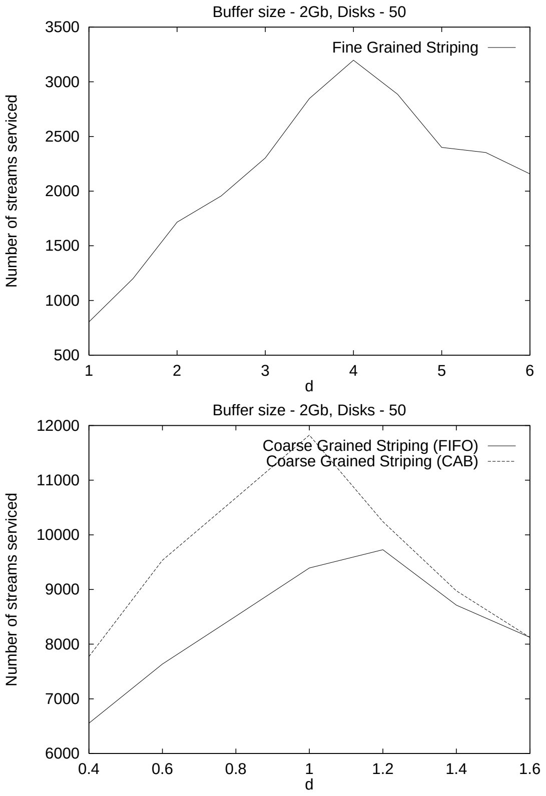

Figure 4: Number of streams v/s d

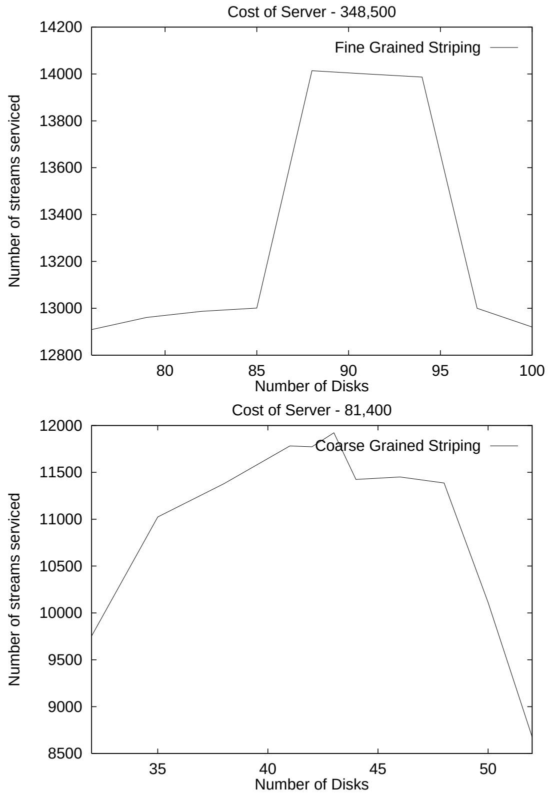

Figure 5: Number of streams v/s number of disks

any value of m, $C(m)$ is greater than $\hat{C}(m)$ ($\lceil \frac{q}{m} \rceil$ is greater than $\frac{q}{m}$), and $\hat{C}(m)$ increases for decreasing values of m below m_{left}, the optimal value of m is an integer between m_{left} and $\lceil m_{calc} \rceil$ such that $C(m)$ is minimum.

The optimal value of m for a server that supports 1000 MPEG-1 streams is 44. The amount of RAM required is 3.08 Gb and the cost of the server is \$81,404. Thus, the cost of the server when coarse-grained striping is used to store videos is almost a quarter of the cost if fine-grained striping were used to support 1000 MPEG-1 streams (see Section 4.2). Also, the value of d is 1.5 Mb and the duration of a round is 1s.

6 Experimental Results

To compare the performance of the schemes proposed in the previous sections, and to validate the optimal values of d and the optimal cost architectures determined in the previous sections, we carried out simulations in a video server environment with multiple disks. Each disk is assumed to employ the C-SCAN scheduling algorithm when retrieving data blocks. The server stores 1000 clips each of length 50 time units. Arrival of client requests into the system is assumed to be Poisson (the mean arrival rate is set at 100 arrivals per unit time), and the choice of the clip for playback by a request is assumed to be random. Each simulation is run for 600 time units and the metric used to evaluate the schemes is the number of clips serviced by the server in the 600 time units. This metric reflects the effectiveness of the schemes in utilizing the memory and disk bandwidth resources (an ineffective scheme would cause the system to be run at lower values of utilization, thereby restricting the entry of clients into the system).

In the first set of simulations, the server is configured with 2 Gb of RAM and 50 disks, and for varying values of d, the number of streams serviced by the fine-grained striping schemes and the FIFO, and CAB coarse-grained striping schemes are determined. In Figure 4, we plot the results of the simulations for the various schemes. From Figure 4, it follows that the CAB coarse-grained striping scheme services the most number of streams (about 12000) when the value of d is 1 Mb. This is the same as the optimal value for d, computed in Section 5.2, for maximizing the number of streams for a server with 2 Gb of RAM and 50 disks. The FIFO scheme services fewer streams (less than 10000) since it does not utilize disk bandwidth as effectively as the CAB scheme. The fine-grained striping scheme performs the worst and services a maximum of 3200 streams when the value of d is 4 Mb which is the same as the optimal value for d computed in Section 4.1. Thus, the simulation results validate the analytically computed value for d in order to maximize the number of concurrent streams.

In the second set of simulations, we only consider the fine-grained striping scheme and the CAB coarse-grained striping scheme. The goal of the simulations is to determine if the optimal architectures for video servers determined (analytically) in Sections 4.2 and 5.3 are indeed optimal. For this purpose, we set the cost of the server to be the costs that were determined in Sections 4.2 and 5.3 in order to support 1000 concurrent MPEG streams (348,500 for the fine-grained striping scheme and 81,400 for the coarse-grained striping scheme). We then consider different server configurations that have the same cost but varying number of disks (the amount of RAM is determined by dividing by the cost of RAM, the difference between the cost of the server and the cost of disks). In Figure 5, we plot the number of streams serviced by the schemes against the number of disks contained in the server. Thus, the simulations yield, for a fixed server cost, the configuration that results in the maximum throughput. For the fine-grained scheme, the number of streams serviced is maximum for 88-94 disks (in Section 4.2, the optimal number of disks was determined to be 88). On the other hand, for the coarse-grained striping scheme, the number of streams serviced is maximum for 43 disks (in Section 5.3, the optimal number of disks was determined to be 44). Thus, the experimental results validate the optimal cost architectures for video servers computed in Sections 4.2 and 5.3.

7 Concluding Remarks

In this paper, we presented schemes based on fine-grained as well as coarse-grained striping for retrieving video data from multiple disks. For the schemes, we computed the optimal amount of data (based on the amount of RAM and the number of disks) to be retrieved for a video during each disk access so that the number of concurrent streams supported by the server is maximized. We also showed how an optimal-cost server that supports a certain pre-specified number of concurrent streams can be architected. Since the coarse-grained striping approach incurs lower latency overhead per disk access than the fine-grained striping approach, the cost of a video server that uses coarse-grained striping is less than the cost of one that uses fine-grained striping. For the coarse-grained striping approach, besides computing the optimal stripe unit size that maximizes the number of concurrent streams that the server can support, we also presented starvation-free schemes for servicing requests that minimize the wasted disk bandwidth.

Acknowledgements: We would like to thank Bob Holt and B. Narendran for sharing with us certain known mathematical results that helped improve our understanding of certain results presented in this paper.

References

[1] *Seagate Product Overview.* Oct. 1993.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. In *Proceedings of ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesota*, pages 79–90, May 1994.

[3] M. S. Chen, D. D. Kandlur, and P. S. Yu. Optimization of the grouped sweeping scheduling (gss) with heterogeneous multimedia streams. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 235–242, Aug. 1993.

[4] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):393–400, June 1994.

[5] P. Chen and D. Patterson. Maximizing throughput in a striped disk array. In *Proceedings of the Seventeenth International Symposium on Computer Architecture, Los Alamitos*, pages 322–331, 1990.

[6] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. A low-cost storage server for movie on demand databases. In *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago*, Sept. 1994.

[7] B. Özden, R. Rastogi, P. Shenoy, and A. Silberschatz. Fault-tolerant architectures for continuous media servers. In *Proceedings of SIGMOD Conference*, 1996.

[8] B. Özden, R. Rastogi, and A. Silberschatz. A framework for the storage and retrieval of continuous media data. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington D.C.*, May 1995.

[9] B. Özden, R. Rastogi, A. Silberschatz, and C. Martin. Demand paging for movie-on-demand servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington D.C.*, May 1995.

[10] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks(RAID). In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data*, 1988.

[11] A. L. N. Reddy and J. C. Wyllie. I/O issues in a multimedia system. *Computer*, 27(3):69–74, Mar. 1994.

[12] A. Silberschatz and P. Galvin. *Operating System Concepts.* Addison-Wesley, 1994.

[13] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A disk storage system for video and audio files. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 393–400, Aug. 1993.