PASTENSE: a Fast Start-up Algorithm for Scalable Video Libraries

Stavros Harizopoulos and Garth A. Gibson

March 2001
CMU-CS-01-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Striping video clip data over many physical resources (typically disk drives) balances video server load with less data replication. Current striped video delivery algorithms can have high start-up latency if the load is high. We propose a new, fast start-up algorithm, PASTENSE. This algorithm minimizes start-up latency by using aggressive prefetching to exploit disk idle time, and using available RAM to dynamically optimize the newly requested video's schedule. Our proposed method (a) does not require changes in the existing striped data placement (b) it never performs worse than alternate designs and (c) it achieves significant benefits: up to 9 times faster start-up times for high loads.

Email: {stavros, garth}@cs.cmu.edu

This research was partially sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. Stavros Harizopoulos is partially sponsored by a Lilian Voudouri Foundation Fellowship and gratefully acknowledges their support. We are also indebted to the member companies of the Parallel Data Consortium for their generous contributions. At the time of this writing, these companies include EMC Corporation Hewlett-Packard Labs, Hitachi, IBM, Infineon Technologies, Intel Corporation, LSI Logic, Lucent Technologies, Network Appliance, PANASAS, Inc., Platys Communications, Quantum Corporation, Seagate Technology, Sun Microsystems, Veritas Software Corporation and 3Com Corporation.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any supporting organization or the U.S. Government.

Keywords: Video servers, scheduling, prefetching, start-up latency, scalability, striping.

1 Introduction

Video server architectures emerged as a significant field of multimedia research over the past decade and gave support to numerous commercial services. Since the introduction of the MPEG video compression standard in 1991 [Gall91], subsequent technological advancements in CPU, I/O, and network performance, and the drop in cost of memory hierarchies, have allowed for the efficient storage of videos into non-volatile massive storage media (disk drives, tapes and optical disks) and their real-time retrieval and transmission over communication networks. Video (and in the same respect, audio) is commonly referred as *continuous media* data in order to differentiate from other multimedia data such as images and hypertext. The continuous media clips need to be delivered to the clients at a specific data rate. Failure to meet this constraint leads to degradation of the quality of service.

The typical model of a video server involves a large, disk-based storage architecture for permanently storing the video data and dedicated RAM space for buffering the retrieved data before transmission. Other design alternatives include an additional tertiary storage hierarchy consisting of a robotic tape or optical disk library from which the popular media streams are extracted and written onto disks periodically. Once a video request arrives, the server performs a simple admission policy based on the available resources and adds the newly admitted stream to the service list. The video server "talks" to the client and transmits the data through the appropriate higher level protocols (such as RTSP, Real Time Streaming Protocol, and RTP, Real-Time Protocol); a network resource allocation scheme might be used prior to any transmission. Data belonging to that stream need to be retrieved from the disks at the appropriate video frame rate, or display anomalies, commonly referred to as "glitches" or "hiccups", will occur. The main memory is used as a transmission buffer for the retrieved data, and also for some extra buffering in order to cope with disk time retrieval variations. Once the video streaming starts, the server uses these memory buffers to guarantee the uninterrupted delivery of the data.

The two principal performance metrics of interest are: (i) the *throughput*, defined as the maximum number of concurrently supported clients, and (ii) the *start-up latency,* defined as the time elapsed from the time the user places the video request until the first frame is displayed. The latter includes the MPEG decoder buffering, the network trip time and the server retrieval time. In this technical report we focus on striped video delivery algorithms which are known to scale well and balance load with less data replication; we analyze the problem of server-added latency under the presence of high load and try to solve it. In Section 2 and 3, we discuss related research efforts and introduce the problem of increased start-up latencies under high loads. Then, in Section 4 we propose a new algorithm, PASTENSE, which applies careful prefetching to existing striped data placement and minimizes the start-up latency for a newly admitted stream. Section 5 carries out a theoretical analysis, and Section 6 presents our analytical and simulation results. We conclude in Section 7.

2 Background

Building a video server involves many design decisions. Most systems adopt the notion of a video *block* (or continuous media block) for purposes of video file organization. A video block is a basic unit for disk storage and retrieval. It is stored contiguously on a disk's surface (i.e. its retrieval involves only one disk head seek), and it sustains video playback for a fixed time interval. During that interval (also called a *Slot*, or *Round*, or *Cycle*), the server reads a block from each admitted stream and transmits it to the client [Reddy94]. Blocks belonging to the same video may also be stored contiguously or in any other way on: (a) just one disk (*all-in-one* or *Independent Disks* design [Chang97a]), or (b) striped with a stripe unit of multiple blocks on all or some of the server disks (*striping design*) [Berson94] [Ozden96]. Within a slot, different disk scheduling policies may be invoked, in order to efficiently accommodate as many retrievals as possible [Yu93].

The server's throughput depends on the ratio $\frac{TotalSeekDelay}{TotalTransferTime}$, on each disk, for every round. *TotalSeekDelay* is the total time spent over a large interval by the disk head seeking and positioning before transferring any requested block. *TotalTransferTime* is the total time spent by the disk, over the same interval, transferring video data. The lower this ratio is, the higher the disk utilization, and thus, the higher the number of concurrently supported streams. *TotalSeekDelay* can be reduced by using a SCAN-like (elevator) disk scheduling policy over the entire block list for a given slot. The *TotalTransferTime* can be increased by choosing a larger round duration, and thus, a larger block size. The trade-off for both techniques is increased buffer space and higher start-up times. For example, a SCAN-like scheduling policy needs two slot-sized buffers in main memory in order to guarantee the uninterrupted data delivery to the client. A retrieved block for a given stream is stored in the buffer and starts transmission sometime after the beginning of the next round. By the end of that next round, the next block retrieval for that stream is guaranteed to have taken place preventing buffer underflow (i.e. before the end of the new round); the new block is stored on the other half of the buffer. Note that this way, there is a minimum server-side start-up latency on average of 1.5 rounds time (half a round on average until a new stream is scheduled for the next round, and one round until the first half of the double buffer becomes full).

A compromise between memory requirements and disk utilization can be achieved by efficiently scheduling block reads within groups and serving these groups in fixed order within each round (Grouped Sweeping Scheduling [Yu93]). Buffer sharing can reduce the memory requirements of a double buffering scheme by 25% [Chang97a]. Modern disks and modern disk controllers may also contribute their large on-board caches, which would stay underutilized otherwise, for prefetching and memory efficient scheduling policies [Triantafillou99]. Besides buffering, the main memory can serve as a cache for frequently requested movies [Dan95]. A highly attractive form of caching for commercial *Video On Demand* servers, is a technique called *stream sharing* [Makaroff95], where new video requests are artificially delayed so that a certain clip can be transmitted to multiple clients at the same time. This way, only one video retrieval per batch of requests is performed and the cost per stream is greatly reduced. The disadvantage in that case is that *VCR operations* (such as pause, fast forward, etc.) are typically restricted and the application of this method in a collection of short movie clips is limited.

A primary concern in systems with multi-disk data layout is load balancing [Soloviev96]. Due to the different popularity of videos within a movie collection, an *Independent Disks* design (the whole video stored in one disk) may impose limitations on the number of concurrent clients. In those designs, the most popular clips need to have multiple replicas in order to accommodate more clients. Striping videos is an inherently scalable approach [Haskin96]: since video clips are striped across all disks, there is no limitation on the number of clients that may view a specific clip (other than the maximum number of clients supported by the system). In this report we examine a typical striping design. We show that load balancing and scalability benefits come at the cost of increased start-up times under high loads, and we propose a new method to minimize the start-up latency.

3 The Problem

Figure 1 depicts a striped video server architecture (shown with just three disks and one "merging" node for illustration purposes). All videos are striped across all disks in a round-robin fashion (as in [Berson94]). The stripe unit has the same size as a video block and sustains playback for the duration of a *slot*. The slot duration is fixed and a typical choice is 1 sec. (see [Shenoy97] for a discussion on choosing the optimal stripe unit size for different workloads). This value is a compromise between (i) disk utilization (as slot duration increases, the disk head spends more time transferring data rather than seeking), (ii) the average time needed to perform a scheduled retrieval within a slot (this is half the slot duration and it adds to the total server-side start-up latency), and (iii) the amount of buffering (higher slot duration results in bigger block sizes and, thus, bigger buffer requirements). We make no assumptions as to how

the disk drives are organized or where they are attached, except that all the disks in the pool can be used to stream data to clients (either through one of the nodes they are attached to, or through one or more central "merging" nodes as shown in Figure 1). This design is representative of systems such as Microsoft's Tiger Video Fileserver [Bolosky96] (a different slot definition is used), the Spiffi Scalable Video Server [Freedman95], and IBM's Tiger Shark [Haskin96] (instead of using a slotted schedule, they employ an earliest-deadline-first scheduling policy).

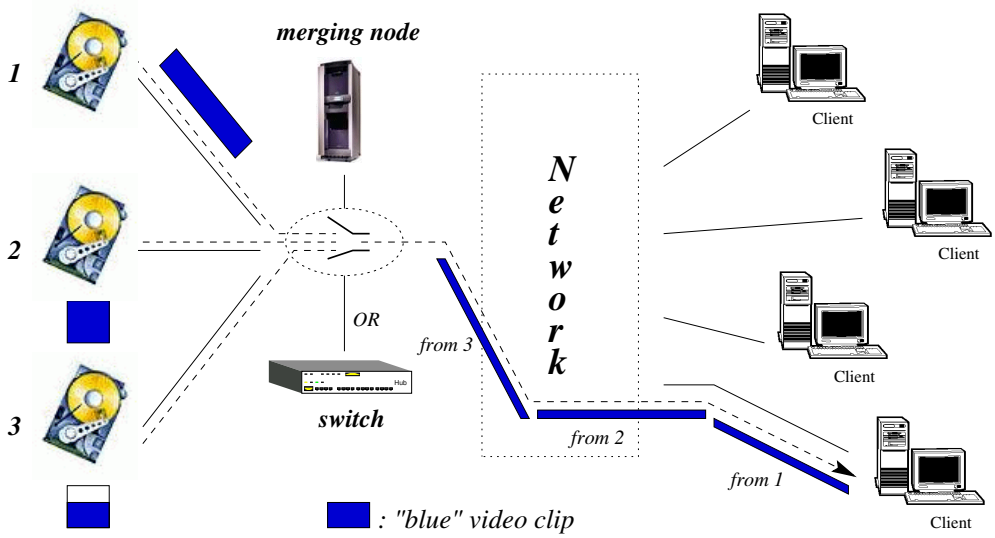

FIGURE 1. System architecture

In Figure 2 we show a schematic representation of the disk schedule for three successive (in terms of striping order) drives. Each rectangle represents a slot, and each colored square is a video block. Blocks of the same color belong to the same movie. The slots are time-aligned for all disks. As time moves down, each disk examines the current slot and performs the necessary retrievals. A clip that is being served by disk 1 in the current slot, will be served by disk 2 in the next slot and so on (after the last disk, the first disk takes again control for that clip). This way, as Figure 2 shows for the "blue" movie, all block retrievals of the same clip happen on a well defined *diagonal* on the disk schedule. As long as there is enough reserved time for a block retrieval in all slots that this diagonal crosses, the client is guaranteed an uninterrupted delivery (this guarantee is from the server point of view; we exclude network considerations and assume a resource reservation scheme is in place [Zhang93]).

A new request is admitted as long as there is at least one free diagonal, and it is placed in the next available diagonal. Within a slot, many blocks belonging to different streams can be retrieved, up to a maximum of *N* blocks, depending on the disk scheduling algorithm (in Figure 2 we have set N to three blocks for simplicity). SCAN-like block scheduling maximizes N, but spends more time buffering before the data streaming can start (a "double buffering" scheme is used for coping with block order variations within different slots). Fixed order scheduling or grouped schemes may result in a lower N for the same slot duration, but they make better use of buffer space and need less time for initial buffering. Note that the notion of the diagonal is independent of the service ordering within a slot. In [Bolosky96], a slot is defined to contain only 1 block. This is a sub-case of Figure 2, where blocks of different clips are retrieved in the same fixed order for every slot and the reserved slot diagonals also specify a relative position within a slot.

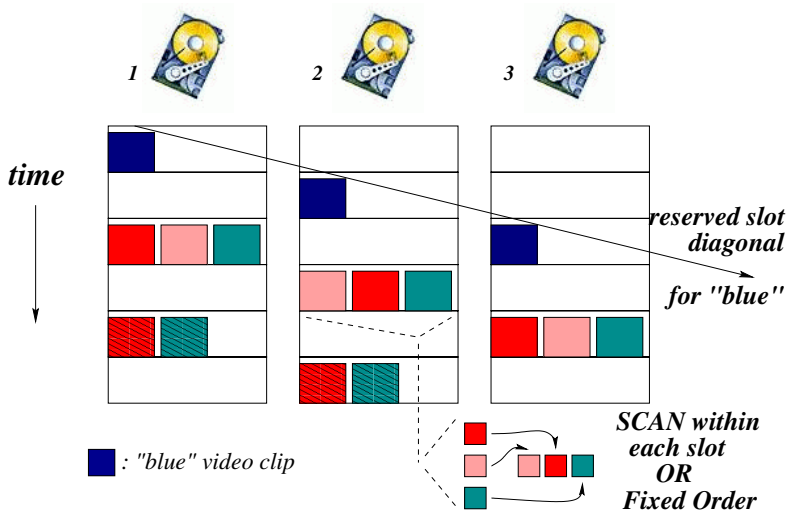

FIGURE 2. Disk scheduling

Although the choice of the retrieval scheme within a slot can add to the start-up latency as much as one or two slots' worth of time (typically 1-2 secs), a more significant start-up delay occurs at high server loads. As more clients are being served, slots get filled. Bursty or clustered client arrivals result in the formation of long chains of filled slots. This way, the available diagonals for a new client can lie far away in future slots and the induced start-up latency could be tens of seconds, depending on the number of drives in the system. This problem is sketched in Figure 3 and is also discussed in [Bolosky96] and [Berson94]. In Figure 3, a new request starting at disk 1 comes when two successive slots are filled; since it "just missed" the previous diagonal, it has to wait for 2 slots for the next available diagonal. For a 100-disk system, server-side start-up delays of 10 - 15 sec. are not unlikely under high loads.

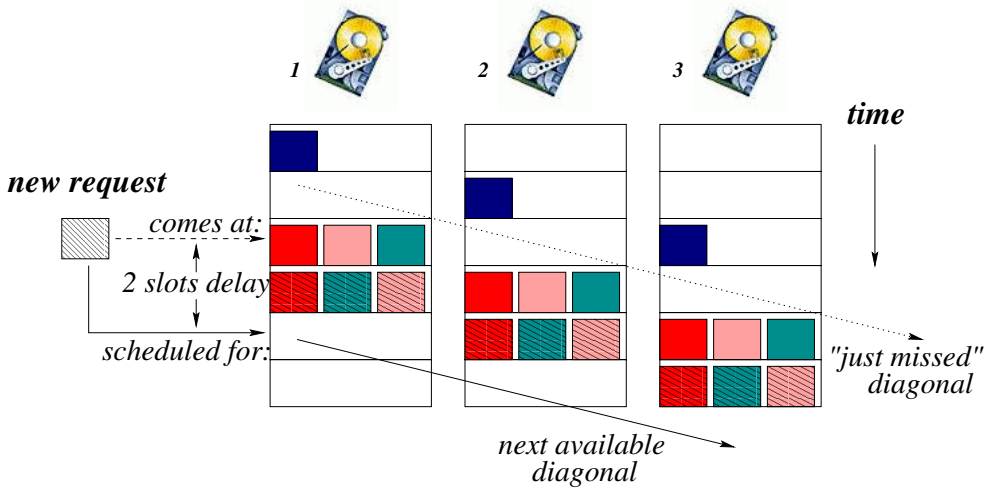

FIGURE 3. The problem

The problem of high start-up times has been addressed for the *Independent Disks* design in [Chang97b] and [Ozden94]. In this design, scheduling and reservations are made on a per-slot basis, so the maximum start-up delay is

bounded by the slot duration, or by twice the slot duration, when SCAN is used. Note that this bound refers to admitted requests; it is likely that new requests will be rejected even though not all disks have reached maximum utilization. On the other hand, in a striping design, where a new request is always admitted as long as the system utilization is not 100%, the worst case start-up time could be orders of magnitude larger than the slot duration. In [Gao99] the authors consider frequently accessed movies, and propose a technique for reducing the client-side start-up latency when a client wants to attend a periodical movie broadcast. To our knowledge, there are no techniques to reduce the start-up times in a striped video server architecture.

4 Proposed method: PASTENSE

As we have already seen, when the striped video server admits a new request, the next available diagonal in the schedule is reserved and used to retrieve the corresponding clip. Note that a "diagonal" in the schedule is a guarantee that all the resources needed for the retrieval of a video will be available in an accurate periodic fashion throughout the whole duration of the clip. The time distance between the current position of the starting disk in schedule and the next available diagonal, along with the time needed for server buffering purposes, make up the server side start-up latency. This delay can be dealt with in two ways: by trying to have an empty diagonal always handy, and, by expediting the buffering time.

The key idea in our method is to place a new request in a *previous* available diagonal (in Figure 3 that would be the "just missed" diagonal). The beginning of such a diagonal lies partly in the past for some disks, hence the name PASTENSE. So while most of the video has an assured retrieval, some disks seem to have already fallen behind in meeting with the specific diagonal's requirements. If we are careful to reserve some extra time within the service of the current slot however, each disk that views this diagonal as in the past can use the spare time to immediately fetch the missing blocks and make up for the "missed" retrievals. This way, we burst fetch the beginning of a clip until we catch up with the chosen diagonal in the present. The number of burst-fetched blocks is equal to the distance in slots of the diagonal chosen in the past and the current slot, for the starting disk. By devoting sufficient disk resources immediately, the video delivery can start right away, after the first block is fetched. PASTENSE is designed to give us this extra time at any point (even if a given slot seems filled) while continuing to allow very high throughput (disk utilization). The algorithm's details follow in the next section.

4.1 The algorithm

Our algorithm is based on extensive prefetching during idle cycles. In addition to buffering in accordance to the media delivery scheme, two new classes of buffers are introduced: the Look Ahead buffer (LA buffer), and the Prefetching buffer (P buffer). During idle slots, the disks try to "look ahead" in the schedule, by prefetching video blocks into the LA buffer, to be used later on. This process is of low priority. This way, during the service of a full slot, it is highly likely that a number of video blocks scheduled for retrieval in the current slot already lie in the cache, and thus, there always exists spare time for immediate disk retrievals. This is shown in Figure 4(a). In the current slot, disk 1 would normally stay idle. Instead, we "look ahead" and prefetch the next block for video 'A'. This block is placed into the LA buffer, and it is going to be delivered to the client in the next slot. Note that in the next slot, disk 1 has created extra room for an extra, out of schedule block retrieval.

When a new request arrives, we find the starting disk in the schedule and try to reserve a past (from the point of current disk's view) diagonal. In Figure 4(b), a new request, 'B', comes and starts from disk 1. The previous available diagonal was one slot back for disk 1, but lies in the current slot for disk 2, and it is still in the future for all other disks. PASTENSE will reserve that retrieval diagonal for the duration of the clip and will also mark that the delivery diagonal for that clip is the one currently starting from disk 1. Disk 1 will burst fetch the first block of 'B', since there now exists some spare time in the current slot, and deliver it to the client. All other disks, including disk 1 for future

periodic retrievals, will retrieve the blocks of 'B' during the retrieval diagonal and store those blocks in the Prefetching buffer until actual delivery. In Figure 4(b), disk 2 fetches the block that belongs to 'B' in the current slot, stores it in the P buffer, and delivers it to the client in the next slot. In general, if the previous available diagonal in the starting disk is k slots in the past, then the next k disks need to be able to burst fetch a total of k video blocks for the new stream.

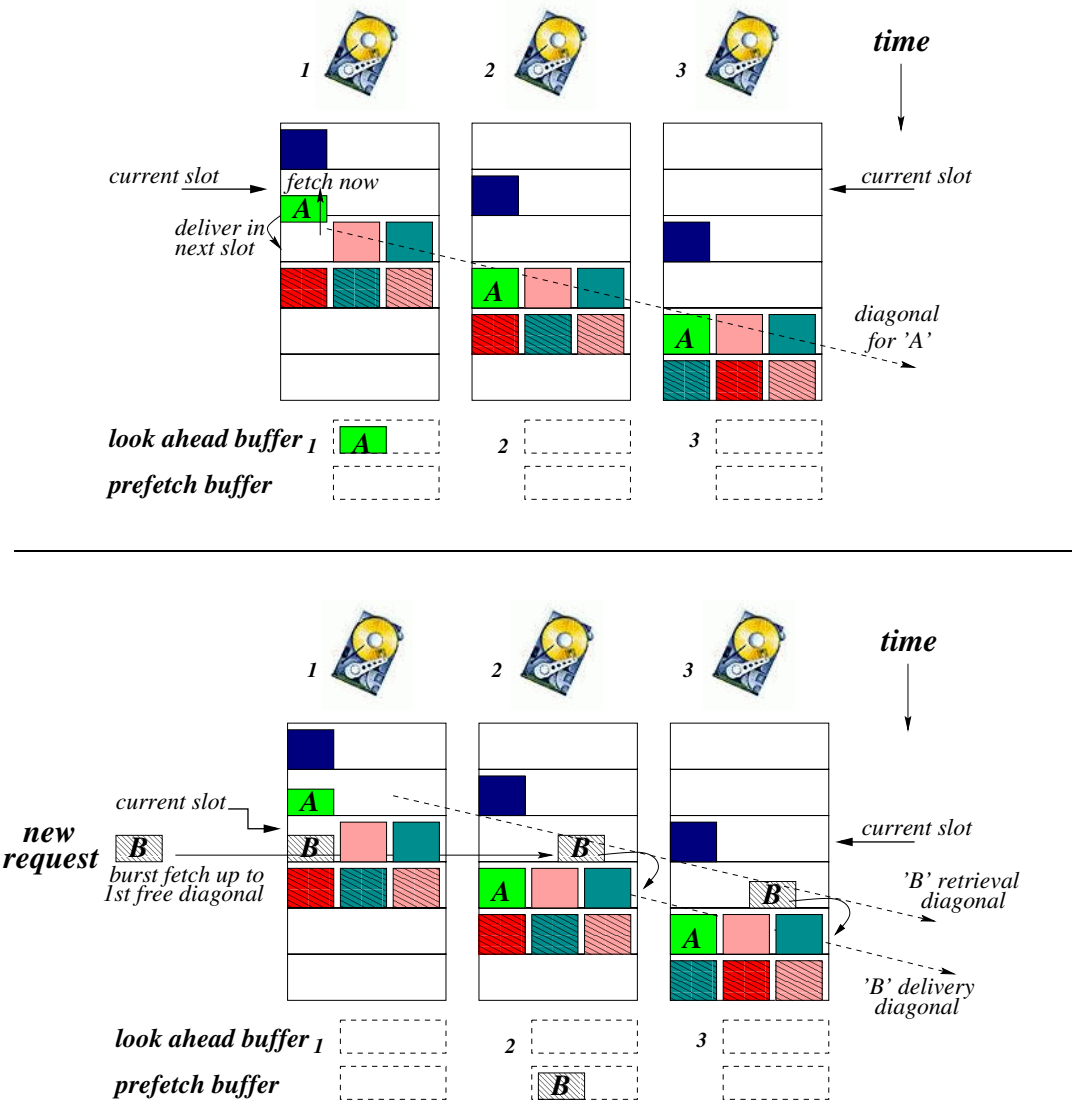

FIGURE 4. (a), (b) Proposed technique: use a past diagonal

Our algorithm checks if all k participating disks can perform this "extra" work by making use of their LA buffer, if necessary. If yes, then we grant the diagonal to the new request. The k disks fetch a block of the requested clip during the current slot and schedule their appropriate delivery. The drives are aware of the time difference between retrieval and transmission, so they use the Prefetching buffer to store that stream's prefetched block, for k slots time. If any of the drives cannot issue the "extra" retrieval, then the algorithm switches back to normal schedule and the request is granted the next available diagonal, with the associated start-up latency.

During idle slots, the drives try to fill up the Look Ahead buffer. This is a best-effort process, and the blocks prefetched and buffered may be replaced with others which are to be used sooner in the future. Each time an "extra" retrieval is issued, the LA buffer is reduced by as many blocks as we would normally retrieve with the default disk scheduling algorithm in the time it took us to perform this "extra" task (this is 1 block for fixed order disk scheduling, but might be more for a SCAN-like scheduling). When the LA buffer empties, server-side start-up latency for new requests may be unavoidable.

The Prefetching (P) buffer increases by 1 block for each new request. Those blocks are only buffered until delivery and cannot be evicted. Once the P buffer fills up, server-side start-up latency is also unavoidable. This buffer is reduced each time a clip with different retrieval and delivery diagonals reaches its end. The Prefetching buffer can also get reduced if we "push" the retrieval diagonal to match the delivery diagonal by secondary "burst" fetching operations once the server load gets low enough to permit this.

5 Theoretical analysis

We use queueing theory to model the video schedule and derive the server-side start-up latency for different client arrival rates. What our mathematical model eventually is going to be is a Continuous Time Markov Chain with finite Capacity and a Load-dependent service rate.

One way to view the "video server" is like $N \times D$ servers, each one serving (playing) a video clip, where N is the maximum number of retrieved blocks during a slot, and D is the number of disks in the system. That is, each diagonal in the slotted schedule is a server in the model. The total number of states in this system is $N \times D$. For the modeling purposes we assume poisson arrivals with rate λ. When the system is full, new clients are rejected (clients do not wait on a queue). The average service time is derived from the average time a video clip lasts, plus the average start up latency. This is also the time between the user's request and the end of displaying the requested video. The slot duration is *T*.

We define P_{AB} to be the probability that a new client will experience start-up latency due to filled slots, that is, when PASTENSE cannot apply. We will later compute this probability. If the average video playing time is $\frac{1}{\mu}$ (chosen from an exponential distribution) and a new client experiences a delay of k slots, or $k \times T$ secs, with probability P_{AB} (the delay is zero slots with probability $1 - P_{AB}$), then each non-rejected client stays in the system for $\frac{1}{\mu} + P_{AB}kT$ secs or, equivalently, service rate is $\frac{\mu}{1 + \mu k T P_{AB}}$ for 1 server. In the i_{th} state, the i clients will be served concurrently, so the departure rate is:

$$\mu(i) = i\frac{\mu}{1 + \mu k(i) T P_{AB}(i)} \qquad (1)$$

We model our system as a Continuous Time Markov Chain with $N \times D$ states (i.e. finite capacity; see Figure 5). Since in the i_{th} state all i clients are served concurrently, we choose to represent our system as a single server system with load dependent service rate. This service rate is given as a function of i, in (1).

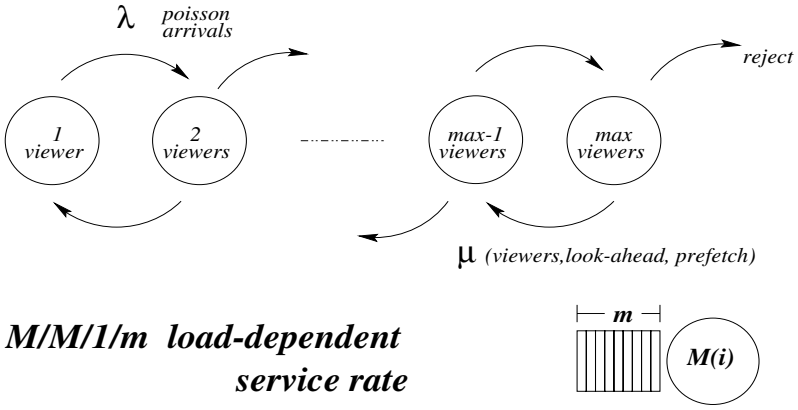

FIGURE 5. M/M/1/m queue with load dependent service

We denote with P_n the probability that there are n clients in the system. The system rejects clients with rate $P_{ND}\lambda$, where P_{ND} is the probability that there are $N \times D$ clients in the system, and thus, the acceptance rate in the system is $\hat{\lambda} = \lambda - P_{ND}\lambda$. Little's Law associates the average number of clients in the system, $\bar{n}$, the average time a client spends in the system, $\bar{\tau}$, and the arrival rate (acceptance rate in our case):

$$\bar{n} = \hat{\lambda}\bar{\tau} \text{ (Little's Law)}$$

Since the average number of clients in the system is $\bar{n} = \sum_{n=1}^{ND} nP_n$, we can derive the average time in system:

$$\bar{\tau} = \frac{\bar{n}}{\lambda - P_{ND}\lambda} = \frac{\sum_{n=1}^{ND} nP_n}{\lambda - P_{ND}\lambda} \qquad (2)$$

But the average duration of a video is $\frac{1}{\mu}$, so the average start-up latency L is:

$$L = \bar{\tau} - \frac{1}{\mu}$$

$$or: \quad L = \frac{\sum_{n=1}^{ND} nP_n}{\lambda - P_{ND}\lambda} - \frac{1}{\mu} \qquad (3)$$

In order to compute the average server-side start-up latency from (3), we need to compute the probability P of each state. By solving the Markov Chain we get:

$$P_n = P_0 \prod_{i=1}^{n} \frac{\lambda}{\mu(i)} \;,\; P_0 = \frac{1}{1 + \sum_{\kappa=1}^{ND} \prod_{i=1}^{\kappa} \frac{\lambda}{\mu(i)}} \text{, and } \quad P_n = \frac{\prod_{i=1}^{n} \frac{\lambda}{\mu(i)}}{1 + \sum_{\kappa=1}^{ND} \prod_{i=1}^{\kappa} \frac{\lambda}{\mu(i)}} \qquad (4a, 4b, 4c)$$

Now we need to compute $\mu(i)$, from (1). Given that there are i clients in the schedule, a new client will experience k slots of delay due to collisions in trying to find the next available diagonal. This case happens with probability P_{AB}, where our algorithm is not able to help, because either the look-ahead buffer is empty, or the prefetch buffer is full (we will compute that probability later). Suppose we view the schedule as a table of ND entries where each client (*or* diagonal) occupies one entry, and an imaginary slot boundary exists every N entries. Then the problem of computing the number of full slots we need to skip to find the next available slot, given a starting point, is isomorphic to computing the number of probes (with respect to slot boundaries) during an insertion in a hash table of size ND, with i entries, where collisions are solved with linear probing. This isomorphism was pointed out in [Bolosky96] and the appropriate analysis has been carried out in [Knuth73]. That analysis gave:

$$k'(i) = \frac{1}{2} \times \left(1 + \sum_{\beta \geq 1} \left[(\beta + 1) \frac{i}{ND} \frac{i-1}{ND} \cdots \frac{i-\beta+1}{ND} \right] \right) ,$$

for the hash insertion problem, and by taking into account the slot boundaries, the expected number of skipped full slots with i clients in the system is:

$$k(i) = \left(k'(i) + \frac{N}{2} \right) \div N \qquad (5)$$

Equation (5) takes into account the effects of clustering. That is, during a sequence of insertions into a hash table, linear probing causes long chains of occupied cells to be created. This analysis does not take into account that keys (clients) may be removed (depart) from the table (schedule). The authors in [Bolosky96] also pointed this out but they didn't provide with a theoretical model neither they studied the accuracy of the formula. In the striped video schedule it is more typical to see small "holes" in highly loaded slots. In [Petterson57], the author analyzed a model assuming uniform distribution of keys into the hash table, called uniform hashing. For our case, this analysis gives:

$$k(i) = \left(\frac{ND+1}{ND-i+1} + \frac{N}{2} \right) \div N \qquad (6)$$

In reality, although client arrivals can be clustered and the policy of looking for the next available diagonal is prone to creating chains of filled slots, when a client (or key) leaves the system, it does so in a random, uniform way with respect to its diagonal position. Thus, formula (6) seems more reasonable for describing the video schedule. In the next section we show that simulation results, for exponential arrivals, suggest that the uniform hashing formula (6) is more accurate than the clustering one, (5).

As we have already mentioned, the probability that our algorithm will not be able to provide with instant playback is P_{AB}. We are now ready to define this probability as: $P_{AB} = P_A + P_B - P_A P_B$, where P_A is the probability that the look-ahead buffer is empty and P_B is the probability that the prefetch buffer is full.

A new arrival in the system will cause an increase in the prefetch buffer by *k(i)* blocks (although all *D* disks will need to prefetch a block of the video, they only keep it in the P buffer for *k(i)* slots; thus, at any time, there are *k(i)* blocks of a pushed back video in the prefetch buffer). When a video clip ends, the P buffer is reduced by the same amount again, *k(i)* blocks. Since the ratio of the arrival rate in the P buffer to the departure rate is the same as in our basic M/M/1/m queue, if we were to approximate P buffer usage as a queue, we could choose a total buffer size of ND blocks or, simply, N blocks per disk, and this way the P buffer would never overflow. Note that N is an upper bound for a safe size for P buffer, since in practice we also anticipate a reduction when we push a retrieval diagonal down closer to the delivery diagonal, when the load becomes less.

With a sufficiently large P buffer, P_{AB} becomes simply P_A. We now need to compute the rates at which the LA buffer gets filled or reduced. When i clients are in the system, a slot is occupied with an average of: $0 \le \frac{i}{D} \le N$ requests. That means, that the LA buffer has a chance of adding $N - \frac{i}{D}$ blocks every T seconds (slot duration), in each disk. Thus, totally, the LA buffer is increased with a rate of $\lambda_A = \frac{ND - i}{TD} \times D$, or, simply:

$$\lambda_A = \frac{ND - i}{T} \qquad (7)$$

For all disks, the LA buffer is reduced on each arrival by the number of consumed blocks in order to perform a random block retrieval, *d*, times the number of random retrieved blocks (these blocks correspond to the time difference between the retrieval and the delivery diagonal and are equal to *k(i)*):

$$\mu_A = \lambda \times d \times k(i) \qquad (8)$$

Given the rate the LA buffer gets increased or reduced, we can compute the departure rate $\mu(i)$ for the system. This is $\mu(i) = i \times \mu$ when $\lambda_A > \mu_A$ (the buffer is increased overall). When $\lambda_A < \mu_A$, the LA buffer will be non-empty $\frac{\lambda_A}{\mu_A}$ of the time, and empty, $1 - \frac{\lambda_A}{\mu_A}$ of the time. When the LA buffer is empty, our algorithm cannot apply and thus, a delay of $k(i) \times T$ secs adds to the video playing time, $\frac{1}{\mu}$. So, the departure rate for the system can be written as:

$$\mu(i) = \frac{i \times \mu}{1 + \mu k(i) T \times max\left(0, 1 - \frac{\lambda_A}{\mu_A}\right)} \qquad (9)$$

The average server-side start-up latency can be computed by plugging the above equations into equation (3). This analysis applies for the baseline schedule if we set P_{AB}, the probability that thePASTENSE technique doesn't apply, equal to 1. Note that for the PASTENSE algorithm, this model doesn't specify the exact amount of needed buffer space for the LA and P buffers. For the LA buffer, since the analysis is based on the rate it grows/shrinks, rather on the size, even a very small size of 1 block can be used and demonstrate the predicted benefits. The P buffer was assumed not to be the bottleneck, and thus a size of N was chosen. In practice, since the P buffer can be greatly compacted, we show that with a modest amount of memory, PASTENSE is able to perform very well.

6 Experimentation

In order to evaluate PASTENSE we set up a simulation environment for the video schedule, similar to the theoretical model. The simulated schedule works with exponentially distributed service times (movie lengths) and client interarrival times. We assume a slot duration of 1 sec. and a minimum of 0.5 sec. start-up time for a new stream. The start-up latency will build up from that value, as slots get filled. The starting disks for different movies are uniformly distributed, in order to achieve better load balancing. Without loss of generality, we set N, the maximum number of retrieved blocks within a slot, to 10 (constant), for all subsequent experiments. We vary the server load, the number of disks in the system and the amount of extra memory given to PASTENSE.

The baseline schedule is the same as PASTENSE with zero extra buffer space. We evaluate a version of PASTENSE where the two extra classes of buffers, Look Ahead (LA) and Prefetching (P), are kept separately. Since these

two classes of buffers grow/shrink in opposite directions, less total memory should be needed by combining them. Note also that the LA buffer performs a low priority, “best effort”- like block fetching, and could thus make use of system memory reserved for video buffering, when the utilization is less than 100%. Instead, the only optimization we use is to dynamically push retrieval diagonals closer to delivery diagonals, as empty slots become available.

6.1 Theoretical model accuracy

In this experiment we test which formula for computing the average number of successive filled slots is more accurate (the one pointed out in [Bolosky96] vs. uniform hashing). For a fixed server load of 96% disk utilization and different number of disks, we plot the average start-up latency derived from the two theoretical models and the simulation, for both the baseline schedule and PASTENSE. The result is shown in Figures 6(a), (b). We see that the theoretical model based on the uniform hashing formula is very close to the simulation results for both algorithms, whereas the model based on the first formula diverges as the number of disks in the system increases. For the PASTENSE simulation we used a minimal buffer size of 1 block for the LA buffer and we removed the P buffer bottleneck, in order to be closer to the model. Based on these results, we prefer the uniform hashing model. In the next Section, we test PASTENSE under memory constraints.

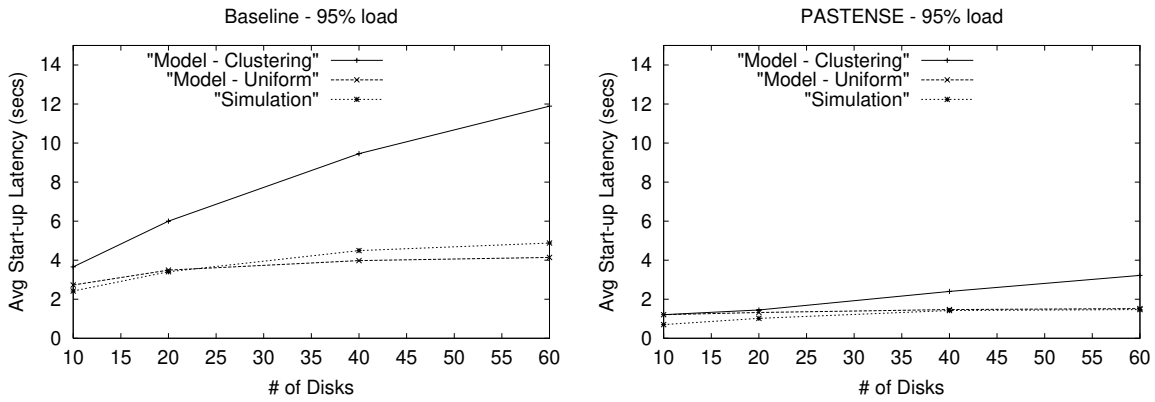

FIGURE 6. (a), (b) Comparison of models and simulation under fixed system load

6.2 Start-up latency versus server load

For this experiment, we take a server configuration with 100 disks and compare the baseline schedule (or PASTENSE 0, zero extra RAM) with 3 different memory setups of PASTENSE: (i) 20% of N blocks, or 2 blocks per disk extra RAM, (ii) 40% of N blocks, or 4 blocks extra RAM, and (iii) 100% of N or 10 blocks. These percentages also correspond to the total extra memory needed by PASTENSE if a fixed order disk scheduling is used, but they are reduced by half, when SCAN is used. For each setup, we choose an optimal split of the available RAM into the two buffers, LA and P. In Figure 7(a) we plot the average start-up latency versus the server load, for high loads (greater than 80%), and average movie length of 15 minutes. Figure 7(b) shows the same experiment for an average movie duration of 1.5 hours.

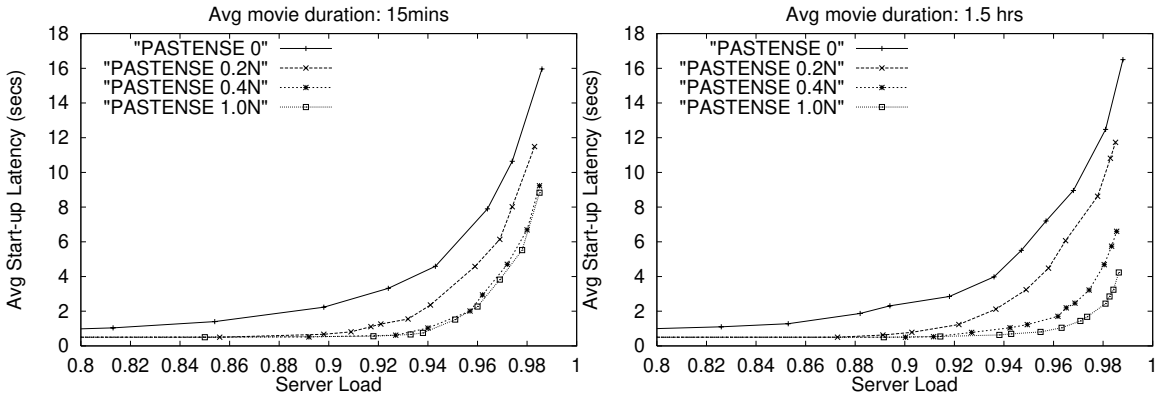

FIGURE 7. (a), (b) Start-up latency versus load for average movie lengths 15mins, 1.5hrs

We see that PASTENSE "pulls" the knee of the start-up delay curve closer to the bottom-right corner, which corresponds to higher loads and faster start-up times. Our technique is able to deliver better start-up times, up to 2 times faster, even with a small amount of extra RAM (2 blocks per disk, for PASTENSE 0.2N), and up to 9 times faster times when 10 blocks of extra RAM are available. For relatively short average movie length (15 minutes), we see that PASTENSE reaches its maximum performance when 40% of N blocks are available, and thus, we don't need to use more memory. However, for longer average movie lengths, we see that PASTENSE is able to perform better with more memory. This is because longer service times cause less variability in the schedule and, consequently, better utilization of thePASTENSE buffers.

In Figures 8(a), (b) we plot again the average start-up latency versus server load, for the baseline schedule and one PASTENSE setup, 0.4N, but this time we show multiple curves, each corresponding to a different number of disks in the system. The "pulled" curve knees are also apparent in Figure 8, for PASTENSE.

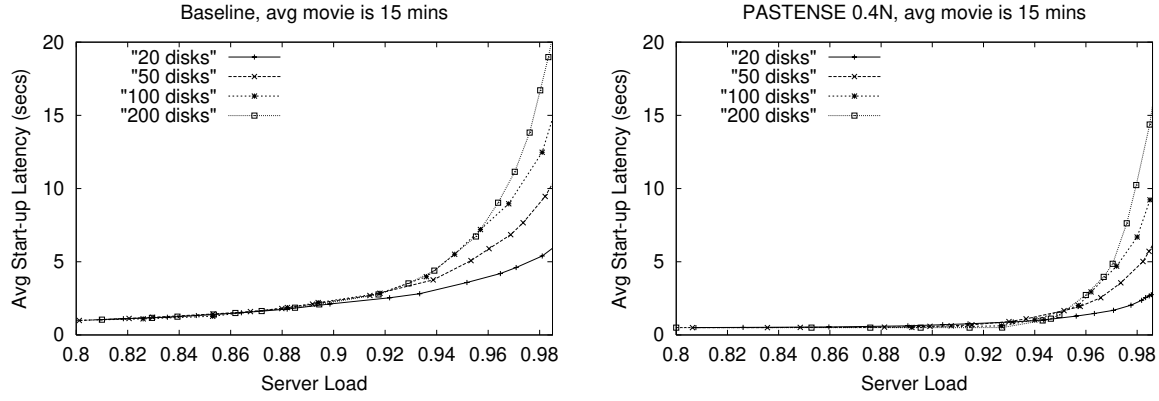

FIGURE 8. (a), (b) Start-up latency versus load under scaling, for baseline andPASTENSE

6.3 Effect of movie length

In Figure 9 we show how the average length of the movies stored in the video server affects the average start-up latency, for a constant, high load of 96%. We can see that the baseline server is rather insensitive to different movie

lengths. PASTENSE, on the other hand, performs better for a mix of short and long clips and its benefits start reducing for average length values less than 20 minutes. This is because higher frequency of arrivals / departures hurts the efficiency of the prefetching done by the Look Ahead buffer. But still, even for clip collections with an average duration of 5 minutes, PASTENSE with a modest amount of extra memory improves start-up times by a factor of 2, when compared to the baseline. We also see that different PASTENSE configurations don't affect this trend.

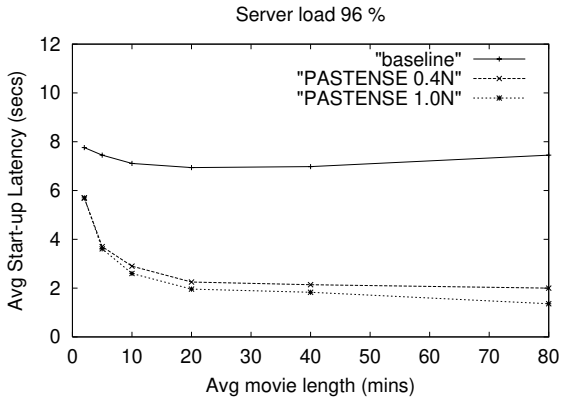

FIGURE 9. Effect of average movie length for constant load

7 Conclusions

In this technical report we presented PASTENSE, a new, fast start-up algorithm for optimizing the video schedule in striped video server architectures. Striped video delivery algorithms exhibit highly desirable scaling and load balancing properties, and have been demonstrated commercially in [Bolosky96]. Their drawback is the potentially large start-up latency under high loads.

We provided an accurate analytical model to study this problem. We proposed a new method which takes advantage of extra memory and achieves up to 2-4 times smaller start-up times with modest amounts of extra RAM, and up to 9 times when using more memory. PASTENSE operates with existing striped data placement and it never loses to the baseline schedule. It is also orthogonal to the block retrieval scheme, thus it doesn't affect the maximum throughput of the server. In our experimental section, we showed that our theoretical analysis closely models both the baseline schedule and our method's behavior. For our proposed method, we showed that relatively little memory can offer big gains, which increase as the server's load increases.

Acknowledgements

We would like to thank Mor Harchol-Balter for her help with the analytical model, and Christos Faloutsos for his many comments on earlier drafts of the report.

References

[Berson94] S. Berson, S. Ghandeharizadeh, R.R. Muntz and X. Ju. "Staggered Striping in Multimedia Information Systems" In SIGMOD'94, Minneapolis, Minnesota, pp. 79-90, 1994.

[Bolosky96] W. J. Bolosky, J. S. Barrera, III, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold, R. F. Rashid. "The Tiger Video Fileserver" In NOSSDAV'96, April 1996.

[Chang97a] E. Chang and H. Garcia-Molina. "Effective Memory Use in a Media Server" In Proc. of 23rd International Conference on Very Large Data Bases (VLDB), August 1997.

[Chang97b] E. Chang and H. Garcia-Molina. "BubbleUp: Low Latency Fast-Scan for Media Servers" Proceedings of the 5th ACM International Conference on Multimedia, p.87-98, Seattle, November 1997.

[Dan95] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, R. Tewari. "Buffering and Caching in Large-Scale Video Servers", Proceedings IEEE CompCon Spring '95, San Francisco, March 1995.

[Freedman95] Craig S. Freedman and David J. DeWitt. "The SPIFFI Scalable Video-on-Demand System" In SIGMOD'95, San Jose, CA USA, 1995.

[Gall91] Didier Le Gall. "MPEG: A Video Compression Standard for Multimedia Applications" Communications of the ACM, April 1991, Vol. 34, No. 4.

[Gao99] L. Gao, Z. Zhang, and D. Towsley. " Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams" In 7th ACM Int. Multimedia Conference, Nov. 1999.

[Haskin96] Roger Haskin and Frank Schmuck. "The Tiger Shark File System" Proceedings of IEEE 1996 Spring COMPCON, Santa Clara, CA, Feb. 1996.

[Knuth73] Donald E. Knuth. "The Art of Computer Programming, Volume 3: Sorting and Searching" pages 527-531. Addison-Wesley, 1973.

[Makaroff95] D. Makaroff and R. Ng. "Schemes for Implementing Buffer Sharing in Continuous-Media Systems" Information Systems, 20(6):445-464, 1995.

[Ozden94] B. Ozden, R. Rastogi, and A. Silberschatz. "On the Storage and Retrieval of Continuous Media Data" 3rd International Conference on Knowledge Management, November 1994.

[Ozden96] B. Ozden, R. Rastogi and A. Silberschatz. "Disk Striping in Video Server Environments" In Proc. of the Intern. Conf. on Multimedia Computing and Systems (ICMCS), June 1996.

[Peterson57] W. W. Peterson, IBM J. Research & Development 1 (1957), 135-136.

[Reddy94] A.L.N. Reddy and J.C. Wyllie. "I/O Issues in a Multimedia System" IEEE Computer, March 1994, pp. 69-74.

[Shenoy97] P. J. Shenoy and H. M. Vin. "Efficient Striping Techniques for Multimedia File Servers" In NOSSDAV'97, St Louis, MO, May 1997.

[Soloviev96] V. Soloviev. "Prefetching in Segmented Disk Cache for Multi-Disk Systems" In IOPADS'96: 69-8, 1996.

[Triantafillou99] P. Triantafillou and S. Harizopoulos. "Prefetching into Smart-Disk Caches for High Performance Media Servers" In Proc. of the Intern. Conf. on Multimedia Computing and Systems (ICMCS), June 1999.

[Yu93] P.S. Yu, M.S. Chen and D.D. Kandlur. "Grouped sweeping scheduling for DASD-based multimedia storage management" ACM Multimedia Systems, 1(3): 99-109, 1993.

[Zhang93] L. Zhang and S. Deering and D. Estrin and S. Shenker and D. Zappala, "RSVP: A New Resource Reservation Protocol", IEEE Communication Magazine, 31(9):8-18, Sept. 1993.