Prefetching into Smart-Disk Caches for High Performance Media Servers

Peter Triantafillou
Department of Computer Engineering,
Technical University of Crete, Chania,
Greece and
Computer Technology Institute, Patras
peter@ced.tuc.gr

Stavros Harizopoulos
Computer Science Department
Carnegie Mellon University, U.S.A
Stavros.Harizopoulos@cs.cmu.edu

Abstract

This paper presents techniques, which exploit recent magnetic disk-drive technological developments (such as the existence of embedded drive-level caches and powerful controllers, and the ever-increasing transfer rates). It contributes prefetching techniques into host- and drive-level caches to improve the maximum number of continuous data streams that a drive can support. We show how our techniques can achieve significant performance improvements while guaranteeing the uninterrupted display of the continuous data. In addition, despite our techniques' utilization of drive-level caches, the performance improvements do not come at the expense of additional cache memory (at the host and/or the drive). Given current technology trends, the benefits of our techniques are expected to become even greater.

1. Introduction

While many algorithms for retrieving continuous data from disks have been proposed, their performance is limited mainly by the disk access time which depends on mechanical movements and is not expected to improve rapidly. However, most vendors provide disks with an on-board cache and powerful controllers, whose exploitation can significantly decrease disk access times.

1.1 Multimedia delivery overview

In this study we focus on disk requests that occur from the playback of video and audio. These data must be retrieved from the secondary memory at a specific rate (or else "hiccups" or "glitches" occur). The usual model of a continuous media server involves the use of magnetic disk from where the requests are served. To ensure the continuous playback for a data stream, typically, a *double buffer* in a main memory cache is needed. The data read from disk fill one half of this buffer, while at the same time the data previously stored from disk into the other half of the buffer, are consumed (i.e., video plays). When the consumption of these data ends, a switch occurs; the full half of the buffer is used for playback and the empty one for storing. As multiple streams must be supported concurrently, the media server typically services these streams in *rounds*. During a *round* one segment (block) for each stream is read and each block contains enough data to sustain the playback for the duration of the *round* [2].

1.2 Disk-related Technological Developments, Trends, and Research Opportunities.

A critical development has been the incorporation of caches into disk drives. For example, modern drives [5,12] can be bought with up to 4MB of cache. The technology trends currently call for even higher-performance disk controllers (with performance similar to a >200 MHz Pentium) at year 2001. These "embedded CPUs" will have available an "embedded cache" with capacity 32-64MB.

Another trend concerns the dramatic increase in the transfer rates of disk drives, owing mainly to dramatic improvements in the linear storage density and to speed-ups in the revolution speeds of drives. Transfer rates will near 30MB/s by year 2,000. However, the disk head's positioning times will not keep pace. This suggests that clever prefetching strategies become even more important for improving the disk system's performance.

Because of the above trends, the concept of *network attached storage devices* (NASDs) receives increasingly greater attention by researchers and developers [3,7,8]. NASDs are storage devices with powerful controllers and large on-board caches, able to run streamlined versions of storage, file related, and network-related, software protocol stacks. Also, several efforts are exploiting these trends by enabling programs to run at the controller level increasing the performance offered to applications. Examples of such endeavors are the *active disks* and *intelligent disks* projects [3,7,8]. In the same spirit as these efforts in "smart disks" is the work in [6] in which special capabilities possessed by modern disk controllers were exploited to devise drastically more efficient scheduling algorithms for mixed-media workloads.

1.3 The problem

The techniques we present here aim to increase the disk drive's performance for continuous media

0-7695-0253-9/99 $10.00 © 1999 IEEE

applications. Our main performance metric is the *maximum throughput* achievable by a drive, defined as the maximum number of supported streams. The proposed algorithms exploit: (i) the predictability that media streams show, (ii) the drive-level and other higher-level caches, (iii) the powerful controllers which can process application-level information, and (iv) the faster pace of improvement of the transfer time versus that of the disk head positioning time. The proposed algorithms are based on prefetching strategies.

Specifically, disk and higher-level caches will be used to prefetch continuous data segments (of one or more streams) that logically follow their display. Despite the significant portion of time that is lost during prefetching of large CM blocks, we show: First, how to utilize higher-level caches to avoid hiccups caused by the prefetching time overhead. Second, that the throughput is increased as the prefetched requests will be served from the disk's cache, without the overhead needed for positioning onto disk, and third, that we can, at the same time, achieve lower total cache memory requirements. We have conducted analyses and experimentation, which testify for these benefits.

We will concentrate on the performance of a single drive. We assume that each disk is *logically* divided in partitions. Each partition is a group of contiguous tracks. It stores a number of consecutive blocks of a continuous object. For example, if at most 20 videos can be retrieved by a disk's bandwidth, given the size of each video (approx. 3GB for an MPEG-2 90-minute video) currently not all 20 videos can fit in a single disk. So we assume 20 logical partitions, each storing a number of consecutive blocks of a video. We stress that this assumed placement is realistic. In a hierarchical multimedia storage server [10] the next video's batch of segments can be brought in time from tertiary storage. In disk array environments, using striping techniques like the ones suggested in [4,9] yields such a placement. Each disk can be assigned a consecutive portion of a video in a round-robin fashion. Each disk then stores a large number of contiguous video portions (e.g., containing tens of blocks), for a large number of videos (such as a few hundred videos). Our problem is to develop techniques, which increase the number of requests that each such disk can support.

2. Proposed Algorithms

Four new techniques for retrieving CM data from disks are presented. Data segments are prefetched into disk and host-level caches. The algorithms reduce the disk access overhead more than it is feasible with conventional policies, while avoiding glitches and reducing start-up latencies.

2.1 The sweep & prefetch (S&P) technique

Scheme Sweep is the well-known scheme that reduces disk seek overhead to improve throughput. During a round, Sweep reads each stream's next segment using SCAN. Double buffers in main memory are needed to avoid glitches.

For a given duration of a round there is an upper bound in the number of streams, N, that can be supported by a disk. This number is mainly dependent on two parameters. The total positioning overhead (i.e., the time needed for the disk head to be positioned on the beginning of a segment) and the total transfer time (i.e., the time needed for the disk data to be transferred to main memory). Both of these delays are experienced when retrieving a data segment, which has not been prefetched and thus this retrieval will be called "*random retrieval*". The number of segments retrieved this way in a round is denoted by the letter v. On the other hand, if the disk head continues reading the following segment and stores it into the disk cache, this "prefetched" segment will be read at no positioning overhead. The number of prefetched segments in a round is denoted by p.

At the end of the round, v streams will have been served (as "random retrievals"), while p segments will be in the cache. In the next round, those p segments will serve p streams at no positioning overhead. Thus there will still be enough time to read from the disk surface a block for v streams and prefetch the next segments for p of these v streams. Because the p segments are read at no positioning overhead, the total number of supported streams $(p+v)$ is now greater.

Figure 1[1] demonstrates S&P. A box refers to a video block. A box under another box stands for the next block -- contiguously placed on disk -- of the same stream. Two adjacent boxes in the same line stand for the blocks of two streams that will be scheduled within a round in the same order as they appear (i.e., in SCAN order).

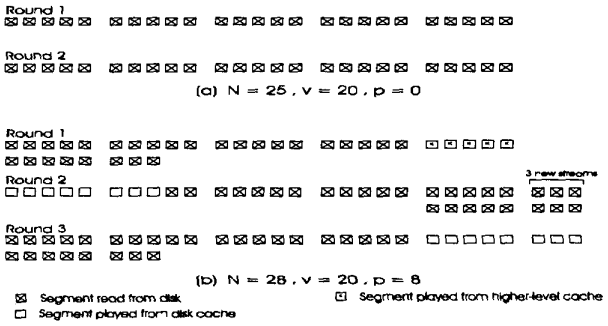

Figure 1: Throughput increment via prefetching

[1] We consider a disk drive with 10 MB/sec transfer rate and a seek and rotational overhead of 15 ms. Also we assume a constant display rate of 2 Mbps and round length of 1 sec. Thus, the size of a segment will be 250 KB. At most 25 segments can be retrieved within a round. while The ratio of randomly retrieved segments that can be "exchanged" with prefetched segments, is 5 / 8 (40 ms and 25 ms retrieval times).

Figure 1(a) introduces the maximum number of that can be supported with Sweep. Then, in Figure 1(b), the prefetching of 8 segments -- one for each of 8 streams -- is used to increase the number of streams from 25 to 28. In each round there are 20 random retrievals and 8 prefetches. The reads from the disk cache to the host cache go on in parallel to the disk accesses $(p+v)$. In round 1, the last 5 streams are supported from an additional host-level cache. These streams will not suffer a glitch. When the requests for these 5 streams arrive, along with the first two blocks for each stream needed by the "double-buffering" scheme, the third block is deposited into the host's cache. The display of these 5 streams is delayed until all three blocks have been retrieved from the disk's surface. Thus, the host-level cache permits not to issue disk retrieval requests for these 5 streams in some round, without these streams experiencing any glitches. The host's extra buffer for this "triple buffering" is no longer needed after round 2 in Figure 1(b).

Scheme S&P can be fully parameterized with v and p, always at the expense of disk cache memory size. Figure 2 shows an alternative combination of values for v and p that results into higher throughput. By prefetching 15 segments, the maximum throughput is increased from 25 to 30. This is the upper bound of the maximum throughput that S&P can achieve in this example. In section 2.3 we will overcome this limitation.

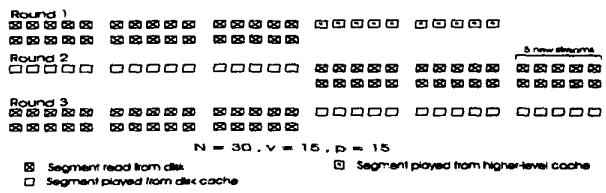

Figure 2: Further throughput increment

2.2 Gradual Prefetching

Scheme S&P, as presented in section 2.1, starts servicing the streams without prefetching until the maximum number of streams are served. Then it switches into the "prefetch" mode. However, it has the drawbacks of requiring three cache buffers for each stream, whose block will be "skipped" in some round and the associated extra start-up latency.

In order to avoid these drawbacks, the S&P scheme can be used all the time. The number of streams that are prefetched at any time will be half of all streams supported. This implies that in every two newly admitted streams for one of them there will always be prefetching of its next segment. This mode will be called Gradual Prefetching. When the number of supported streams reaches the maximum, then the Gradual Prefetching Scheme behaves as the S&P scheme shown in Section 2.1. At this time, half of the streams will have their next blocks already in the disk cache and so if they are skipped in the next round, these streams will not experience hiccups.

2.3 Multi-Round Sweep & Prefetch (MS&P)

Further increase in the maximum throughput is possible by prefetching more segments for some streams. Figure 3 demonstrates 2-level and 3-level prefetches. The throughput is increased from 25 to 34 at the expense of required cache size. Gradual prefetching can be applied to this scheme, once the depth of prefetching is known (this depends directly on p). Figure 3 shows that the hiccup-free display of all streams is ensured. But with MS&P, a technique ensuring that prefetches of different streams do not interfere, overwriting each other's cached blocks, is required. This technique, controlling for which streams prefetching is done, is shown in Figure 4.

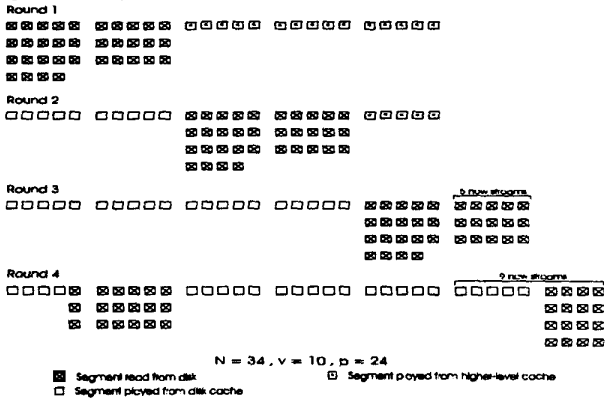

Figure 3: Throughput increment with MS&P.

In MS&P, since multiple segments are prefetched for a stream, the next stream S_i to be read from the disk surface will be separated (in terms of data placement onto disk) by stream $S_j, j < i$ previously read from the disk surface, by as many streams as the number of prefetched segments for S_j. This ensures that the scheduled requests for the segments $S_k, j < k < i$ of the streams placed between the two streams S_i, S_j will be satisfied from cache in time that a place in cache will be free for the segments of the stream S_i to be prefetched. (Recall that the scheduler uses a SCAN policy).

When different levels of prefetches occur in MS&P, a similar policy must be followed, but with two different values of the number of skipped streams. One for each of the two groups of streams that have the same number of prefetched segments. In figure 4 the diagram of MS&P in figure 3 is redrawn to exemplify this.

2.4 Group Periodic Multi-round Prefetching (GPMP)

Scheme GPMP introduces the concept of an *epoch*. The time interval of an *epoch* (or *virtual round*) is defined

as the total duration of a fixed number $(u+1)$ of actual rounds.

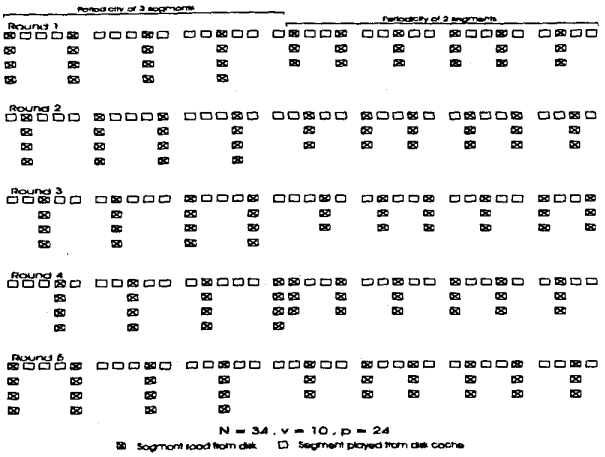

Figure 4: 2-way periodical MS&P

During a round of GPMP all streams will be served. The key idea is that their segments do not have to be retrieved during each round from the disk; some will be found in the cache. Thus, during a round only a group containing a fraction, v, of the streams supported will be randomly retrieved while there will remain enough time within the round for u prefetches for each stream to take place. The letter u is used here for the number of segments prefetched within the same stream.

In the next round, the segments that sustain the playback of the next group, also containing v streams, will be read from the disk surface along with the u prefetched segments for each one of the v streams. All streams will be served during each round because the $N-v$ segments that have not been retrieved from the disk during the current round will be found in the cache, since they had been prefetched in the previous rounds. The u prefetched segments of a stream read during a given round will sustain its playback from the cache for the next u rounds.

The *epoch* consists of $(u+1)$ rounds. During an epoch a complete sweep of the disk is performed. From a disk behavior point of view, GPMP with the epoch looks like the Sweep scheme with a round duration equal to the epoch, but the aggressive prefetching strategy proposed results in a round whose duration is $(u+1)$ times shorter. This strategy also incurs significantly lower memory requirements.

The total number of streams supported under GPMP is $N = v \times (u+1)$. Figure 5 demonstrates GPMP. The maximum number of streams supported at $u = 5$ is 36. ($v = 6$). Gradual Prefetching strategies ensure that during Round 1 in figure 5, the 30 streams not served from disk have their segments already stored in the cache as a result of prefetching during the previous rounds. More specifically, as it is assumed that Round 1 corresponds to the status of an instance of the server working at its maximum throughput, each of the second group of 6 streams in the figure have 1 remaining segment in the cache, each of the third group of 6 streams have 2 remaining segments in the cache, and so on.

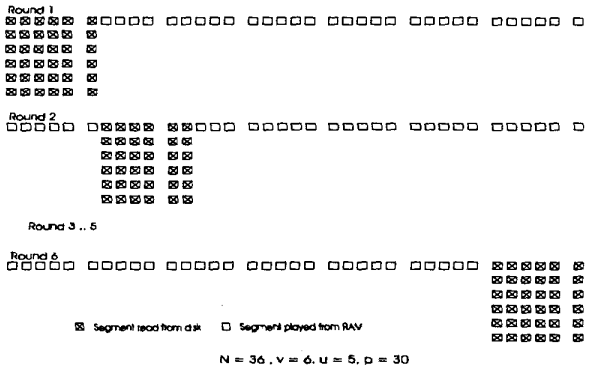

Figure 5: Throughput increase with GPMP

3. Analysis

The parameters used for the analysis, are:
R: the duration of a round; It is constant.
N: the maximum number of streams that can be displayed.
B: the size of a continuous object segment.
p: the total number of prefetched segments.
v: the number of randomly retrieved segments.
u: the number of prefetched segments of the same stream.
Mem: the total size of required system's RAM memory.
dCache: the size of the required disk cache memory.
Transfer: the transfer rate of the disk (disk surface to memory).
Display: the display rate of a stream.
pos(d): computes the positioning overhead (seek plus rotation) of the disk head, given seek distance d.

3.1 Throughput Analysis

We will express the maximum throughput of Sweep and the S&P family of techniques (except GPMP, since it employs a constant round) as a function of R. The round lasts as long as a single segment is displayed. That is:

$$R = B / Display \qquad (1)$$

During a round also, N segments are transferred from disk (or disk cache) to main memory. The delays that each segment experiences are the disk head positioning and disk transfer time for Sweep. In the S&P family of algorithms, only v segments meet these delays. The remaining p segments are played directly from the cache at no time overhead. The time that remains until the completion of the round is used for prefetching p additional segments at the overhead of the disk transfer time only.

In order to ensure continuous playback for all N streams, the worst case positioning overhead must be assumed (disk transfer times do not vary). The most time-consuming sweep would include all N segments separated by equal number of cylinders, or a seek distance of *pos(TotalCyl / N)*. For convenience, this time is called ***POS***. The time needed for a segment to be transferred from disk to memory is equal to $B/Transfer$. As we try to accommodate as many streams as possible during a round, we derive the following equations for R:

$$\text{Sweep:} \quad R = N \times (POS + B/Transfer) \tag{2}$$

$$\text{S\&P family:}^2 \quad R = v \times (POS + B/Transfer) + p \times B/Transfer \tag{3}$$

In the above equation, we assume that the internal transfer rate is equal to *Transfer*.

The number of streams that are supported with S&P algorithms are $N_{S\&P} = v + p$. For GPMP it becomes $N_{MS\&P} = v \times (u+1)$, but by setting $p = v \times u$, the previous relation can be used. It should be reminded here that N is different for different schemes. Equation (3) can be written as:

$$\text{S\&P family:} \quad R = N \times B/Transfer + v \times POS \tag{4}$$

From the above equations we can solve for the size of a segment B as a function of N for both schemes.

$$\text{Sweep:} \quad B = \frac{N \times POS \times Display \times Transfer}{Transfer - N \times Display} \tag{5}$$

$$\text{S\&P family} \quad B = \frac{v \times POS \times Display \times Transfer}{Transfer - N \times Display} \tag{6}$$

From equations (1), (2), and (4) as well as from the relation $N_{S\&P} = v + p$, we can solve for the maximum throughput N supported from each scheme:

$$\text{Sweep:} \quad B = \frac{N \times POS \times Display \times Transfer}{Transfer - N \times Display} \tag{7}$$

$$\text{S\&P family} \quad B = \frac{v \times POS \times Display \times Transfer}{Transfer - N \times Display} \tag{8}$$

From the above equations we can derive the maximum gains that we achieve with the S&P algorithms over Sweep, for a given round R (or a given segment B):

$$N_{S\&P} = N_{Sweep} + N_{Sweep} \times \frac{p \times POS}{R} \tag{9}$$

We can see from the above equation that S&P techniques can further increase the throughput (to an amount tuned by p), while they keep R (and the segment size) constant. If we wanted to increase the throughput of Sweep without these techniques, we would have to make the segments larger and thus extend the round. This would mean higher latency and higher memory requirements. We can also see from equation (9) that the gain of S&P algorithms over Sweep is $p \times POS/R\%$. Please note that the value of p is different for each of the algorithms in the S&P family, as discussed in sections 2.1, 2.2, and 2.3.

3.2 Analytical Results

The values of *Transfer* and *POS* used for the graphs are different from those used in the examples in figures 1-4. An up-to-date disk drive is assumed with average transfer rate of 14 MB/sec and average positioning delay of 11 msec [11].

In this evaluation, techniques Sweep and Fixed-Stretch [1] are considered for comparison with the techniques presented here. This is done because these are representative of two opposite choices in the fundamental tradeoffs: Sweep minimizes seek delays at high memory requirements while Fixed-Stretch minimizes the memory use at the worst seek overhead.

In figure 6, a throughput versus round duration graph has been constructed for schemes Fixed-Stretch, Sweep, S&P, and MS&P (for the last two schemes various cache configurations are considered). The theoretical upper bound of throughput achieved is 56 -- that is, the disk head transfers all data at no positioning overhead. For Fixed-Stretch, a worst case positioning overhead of 22 msec was taken into account, while the other schemes assume an average positioning delay of 8 msec. As it can be seen from the graph, all schemes improve their performance by increasing the duration of the round. This happens as the segment size is proportionally increased resulting in a higher ratio of transferring time to access time (greater disk utilization). The throughput increase is slowed down as the round duration continues to increase, as the positioning overheads are never eliminated. All schemes converge for large round lengths, approaching marginally the upper bound.

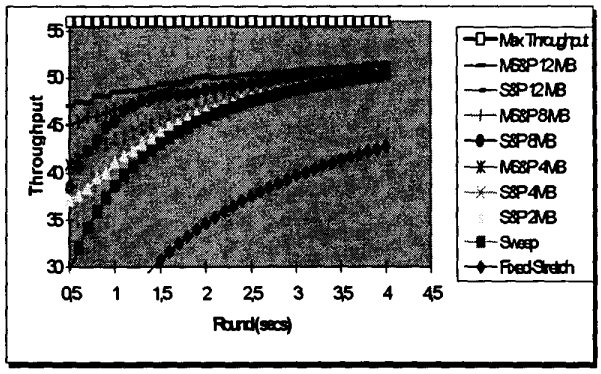

Figure 6: Throughput versus round duration

[2] POS is a function of N and so varies for each scheme. However, typical values of N result into slight differences in POS for the two schemes; therefore, for simplicity, the same POS in calculations is used for both schemes.

It can be seen from figure 6 that the S&P family of algorithms achieve significantly higher throughput than Sweep and Fixed-Stretch especially for short rounds (that also imply short star-up latency). For example, for a round length of 500 msec, MS&P achieves a throughput gain of 50% over Sweep.

As the round increases, MS&P curves meet those of S&P for the same cache sizes. This happens as the segment sizes become large and there is not enough cache for multiple round prefetches. Similarly, as the round decreases, more segments can be prefetched and so S&P does not fully exploit the disk cache.

In the next figure, curves of throughput versus memory are shown (although the relevant analysis is omitted for space reasons). GPMP was plotted assuming a *constant* round of 200 msec. S&P algorithms outperform Sweep and Fixed-Stretch. For GPMP, the memory savings at high throughputs over Sweep are very significant (over 60%). This is explained as follows. In order for Sweep to increase its throughput, it has to continuously increase the round length. This is turn results in greater block sizes, which increase the total memory requirements.

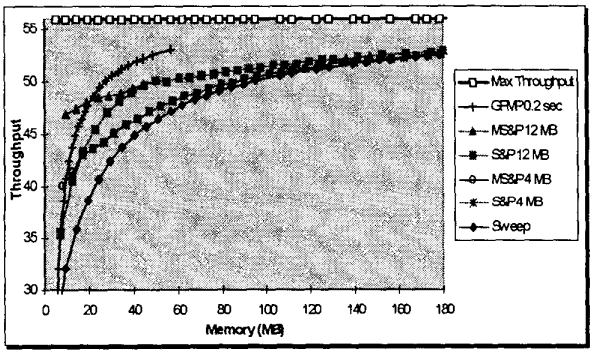

Figure 7: Throughput versus Memory

4. Conclusions

We contributed the S&P technique, which is based on prefetching blocks into disk and host-level caches and improves the disk drive's maximum throughput. The Gradual Prefetching technique avoided the drawbacks of triple-buffering in S&P. MS&P overcame the maximum throughput upper bound of S&P by prefetching more CM data blocks. The GPMP algorithm introduced a round breakdown scheme and an efficient way of organizing streams, which lead to even higher performance.

The central conclusions are that:

- Our prefetching strategies introduce significantly higher disk maximum, up to 60-70% when compared to Sweep.
- In addition, despite the additional disk cache utilized by our strategies, the same investment in total RAM (at the host and/or the disk) for our strategies and Sweep results in higher achievable maximum throughput with our strategies.
- Finally, the current technology trends suggest that for future disk products our techniques will show even better results. Transfer rates will continue to improve at a much faster pace than seek delays and more powerful controllers operating on even bigger embedded caches will appear.

The performance of the proposed schemes was studied through analysis. The validation of the performance benefits claimed by the analysis came through the implementation of the proposed algorithms into a drive accurate simulation model. (The details have been omitted for space reasons; please see [11]). In the future we plan to study the relationships between our techniques and disk scheduling algorithms such as Fixed Stretch [1] and GSS [13].

References

[1] E. Chang and H. Garcia-Molina, "Effective Memory Use in a Media Server (extended version)", Stanford University, Tech. Report SIDL-WP-1996-0050. (http://www-diglib.stanford.edu).

[2] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe, "Multimedia Storage Servers: A Tutorial", IEEE Computer, May 1995, pp.40-49.

[3] K. Keeton, D. A. Patterson, and J. Hellerstein, "A Case for Intelligent Disks (IDISKS)", *SIGMOD Record*, vol. 27, no. 3, August 1998.

[4] B. Ozden, R. Rastogi and A. Silberschatz, "Disk Striping in Video Server Environments", IEEE Intern. Conf. on Multimedia Computing and Systems, June 1996.

[5] `http://www.quantum.com/products/hdd/atlas_10k`.

[6] Y. Rombogiannakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum, "Disk Scheduling for Mixed-Media Workloads in Multimedia Servers", 6th ACM Multimedia Conference, September 1998.

[7] E. Riedel, G. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia Applications", In Proc. of Int. Conf. on VLDB, 1998.

[8] E. Riedel and G. Gibson, "Active Disks – Remote Execution for Network-Attached Storage", Tech. Report, CMU-CS-97-198.

[9] P. Triantafillou and C. Faloutsos. "Overlay Striping and Optimal Parallel I/O in Modern Applications", Parallel Computing, March 1998, (24) pp 21-43.

[10] P. Triantafillou and T. Papadakis, "On-demand Data Elevation in Hierarchical Multimedia Storage Servers", Proc. 23rd Int. Conf. on VLDB, August 97, 226-235.

[11] P. Triantafillou and S. Harizopoulos, "Prefetching into Smart-Disk Caches for High Performance Media Servers", (extended version – available upon request).

[12] http://www.seagate.com/disk/cheetah.

[13] 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.