


Stavros Harizopoulos
Carnegie Mellon University

Costas Harizakis and Peter Triantafillou
Technical University of Crete

 To avoid the throughput limitations of traditional data retrieval algorithms, the authors developed a family of algorithms that exploit emerging smart-disk technologies and increase data throughput on high-performance continuous media servers.

Hierarchical Caching and Prefetching for Continuous Media Servers with Smart Disks

Due to the rapid evolution in high-bandwidth networks and the dramatic increase in CPU performance, I/O systems have become the performance bottleneck for demanding applications. Researchers have proposed many algorithms for retrieving data from disks,¹⁻⁸ but the throughput is limited. The primary

reason for this is disk access time, which depends on slow mechanical movements that are unlikely to speed up in the near future. Storage device vendors are thus offering disks with ever more powerful embedded controllers and increasingly big drive-level caches (see the sidebar “Trends and opportunities in disk technology”). Research efforts, in turn, are directed at developing “smart disks” to exploit these resources and thus improve overall system performance.

This rapidly emerging smart-disk technology introduces additional system resources, such as embedded drive-level caches and powerful controllers. For example, in a general system architecture for a continuous media server, different cache hierarchies coexist with powerful embedded controllers at different points on the data path (see Figure 1). These coexisting elements permit greater parallelism between algorithms operating on distributed caches within the system. More specifically, we can view the CPU and the disk

I/O controllers and SCSI controller as different processing units; the data can flow between the disk caches, the multiple disk controller buffer, and the host cache.

We have developed several algorithms that exploit these coexisting elements in continuous media applications. Our algorithms work in parallel to retrieve, or “prefetch,” data from the disk surface to either the disk or the SCSI controller caches and concurrently transfer the data from the lower cache hierarchies to the host cache. At the same time, the host streams the data from its RAM through the network to the clients. As we will describe, this parallelism—which is mainly based on the intelligent controllers and the different caches—can significantly improve media server performance.

We measure performance in three ways: by the maximum number of continuous data streams that a drive can support, the total RAM size requirements, and the start-up latency. Our caching and prefetching algorithms dramatically improve the num-

Trends and opportunities in disk technology

The incorporation of drive-level caches into disk drives has been critical for improving I/O performance. For example, you can now buy Quantum drives with up to 8 MBytes of cache. When used along with caching and prefetching techniques, these embedded disk caches can significantly increase the drive's performance. Moreover, storage technology for large storage servers lets you introduce caches at multiple levels of the storage hierarchy, including the host's cache, the multidevice controller's cache, a disk array's controller cache, and the drive-level cache.¹ Another significant area of research deals with letting applications pass hints to the underlying operating system (or even the controller). These hints can, for example, describe future access references and thus help create efficient plans for prefetching and caching.

General technology trends call for even higher performance from disk controllers—we expect performance greater than a 200 MHz Pentium by

2001. These “embedded CPUs” will have an “embedded cache” with much greater capacity than what is available today. Another trend is the dramatic increase in disk-drive transfer rates, owing mainly to dramatic improvements in linear storage density, increased revolution speeds, and higher bandwidth I/O buses. The transfer rates have already reached 40 Mbytes per second. However, the expected increase in disk-head positioning time is not expected to keep pace. This suggests that clever prefetching strategies will become even more important for improving the disk system's performance.

Because of these trends, researchers and developers have grown increasingly interested in the concept of *network-attached storage* (NAS) or *network-attached secure disks* (NASDs) (see, for example, www.pdl.cs.cmu.edu/NASD). NASDs are storage devices with powerful controllers, operating on large on-board caches. They can run streamlined

versions of storage, as well as file- and network-related software protocol stacks. Several efforts are attempting to exploit these trends by letting programs run at the controller level and thereby increase the performance that storage devices offer applications. One such endeavor is the *active disks* project, which aims to enable special device operations unavailable with current interfaces.²

References

1. R. Karedla, J.S. Love, and B.G. Wheery, “Cache Strategies to Improve Disk System Performance,” *Computer*, Vol. 27, No. 3, Mar. 1994, pp. 38–46.
2. E. Riedel, G. Gibson, and C. Faloutsos, “Active Storage for Large-Scale Data Mining and Multimedia Applications,” *Proc. Int'l Conf. Very Large Databases (VLDB)*, Morgan Kaufmann, San Francisco, 1998.

ber of streams that a drive can support, with small start-up latencies. In addition, despite using “extra” drive-level caches, these performance improvements do not come at the expense of total additional cache memory (at the host or the drive). Also, given current technology trends, the benefits of our techniques will likely increase over time.

The application domain

A continuous media server offers clients access to media types such as video, audio, and so on. To avoid presentation anomalies, or “glitches,” the media server must retrieve media data continuously from the secondary memory at a specific rate. For example, a media server might involve a tape library, from which it periodically extracts media streams and writes them onto hard disks. The server then serves client requests from the disks. Typically, we require a double buffer in main memory to ensure a data stream's continuous playback. This buffer simultaneously fills and empties: data read from disks fills half the buffer, while the other half is consumed (the video plays). When data consump-

tion ends, a switch occurs and the media server uses the newly full buffer to continue transmitting playback data and uses the empty buffer for storage.

Because multiple streams must be sup-

ported concurrently, the media server typically serves streams in *rounds*,¹ which endure for a given length of time. During a round, the system reads one block from each stream, which is adequate to sustain

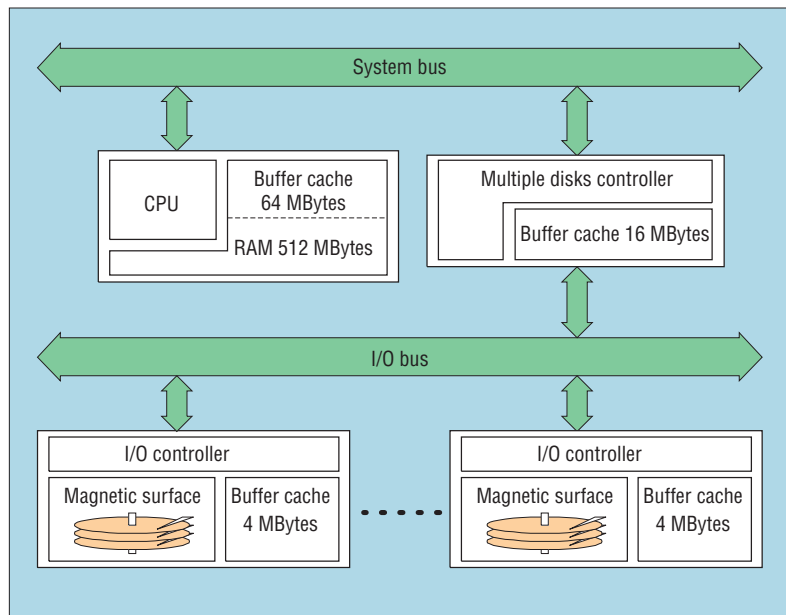


Figure 1. A general system architecture for a continuous media server.

playback for the round's duration. As soon as the clients' requests arrive, a call admission policy decides which requests the server will satisfy immediately, based on the available resources. The server rejects any remaining requests or places them in a queue. A high-level scheduler processes the admitted requests. In our example, each video request that must be served during a round is added to a service list. A resolution mechanism maps each requested video block to many adjacent disk sectors (typically, it maps one video block to a single disk block). The I/O controller routes each newly created disk-block request list to the appropriate disk-drive device. Then, the low-level scheduler schedules the corresponding batch of requests separately for each drive, reducing the disks' head-positioning overhead as much as possible. The disk heads then transfer the requested data from the disk surface to the disk-level buffer cache and through the I/O bus to the server's (host) RAM. Finally, the application running on the continuous media server transfers the data through the network to clients.

Proposed algorithms

Our main performance metric is the *maximum throughput* achievable by a drive, which we define as the maximum number of streams that a drive can support without glitches occurring. Two important factors that limit a media server's cost-performance ratio are the time required to transfer data from the disk to main memory and the cost of the cache memories. Recent research has shown that these two parameters are strongly interrelated and achieving high throughput often comes at a huge memory cost.³ In our method, we increase maximum throughput while keeping round durations constant, with relatively low memory requirements and start-up latency. Thus, for a given maximum throughput, we can actually achieve better memory use and lower latency.

The algorithms we present here exploit several elements of a continuous media server's architecture, including

- the strong predictability of continuous media requests;
- drive-level and other higher level caches; and
- powerful controllers, which can accept application-level hints about the sequential nature of requests.

Our algorithms also take advantage of the trend toward improving performance through faster transfer time versus the relatively weaker gains from improving disk head-positioning time.

To improve the drive's maximum stream throughput, our algorithms use caching and prefetching strategies. More specifically, they use disk and higher level caches to prefetch continuous data blocks (of one or more streams) that logically follow their display. Despite the significant time loss during prefetching of large continuous media blocks, we show here how using higher level caches avoids the glitches caused by prefetching time overhead. Our technique actually increases overall stream throughput, because prefetched requests are served from the disk's cache, without incurring head-positioning overhead. Finally, our techniques offer lower overall cache memory requirements.

In the following examples, we focus on a single drive's performance. We assume that each disk is logically divided in partitions and that each partition is a group of contiguous tracks that stores many consecutive blocks of a continuous media object. For example, if a disk's bandwidth can retrieve a maximum of 40 videos, then given the size of each video (approximately 3 GBytes for an MPEG-2 90-minute video), not all 40 videos will fit in a single disk. Given this, we assume that there are at least 40 logical partitions, each storing several consecutive blocks of a video, and that this placement is realistic. In a hierarchical multimedia storage server, for example, we can bring the next video's batch of blocks in from tertiary storage. As another example, most disk array environments use coarse-grained striping techniques,⁴ which create a placement like the one we assume. In particular, we can assign each array disk a consecutive portion of a video in a round-

robin fashion. After such placement, each disk stores many contiguous video portions (containing tens of blocks, for example) for numerous videos (up to a few hundred). Our problem here is to develop techniques that increase the number of requests that each disk can support.

SWEEP AND PREFETCH

Scheme Sweep is a well-known continuous media-retrieval scheme that improves throughput by reducing disk-seek overhead. During a round, Sweep reads each stream's next block using a SCAN-like scheduling policy.⁷ Double buffers in the main memory ensure continuous playback.

In a given round, a media-server disk can support a specific upper bound in N streams. This number is mainly dependent on two parameters: the total positioning overhead (the time a disk needs to position the disk head on the beginning of a block) and the total transfer time (the time it needs to transfer disk data to main memory). Both of these delays occur when the disk head retrieves a data block that has not been prefetched; we therefore call them *random retrievals*. We denote the number of blocks randomly retrieved in a round by the letter v . The disk head prefetches blocks when it reads adjacent blocks and stores them into the disk cache. Reading prefetched blocks creates no positioning overhead. We denote the number of prefetched blocks in a round by the letter p . If we force the disk head to continue reading the next block of the same stream into the disk cache and do this for p streams, then at the end of the round, the system will have served v streams (as random retrievals), while p blocks will reside in the cache. In the next round, those p blocks will serve p streams. Because reading p blocks creates no positioning overhead, the total number of streams the disk can support is now greater. That is, we have increased maximum throughput, while keeping the round's duration and the block size constant.

Figure 2 demonstrates our Sweep and Prefetch (S&P) technique. Figure 2a shows the maximum number of streams (25 in this example) that Sweep can sup-

port. In Figure 2b, we prefetch eight blocks—one for each of the eight streams—to increase the number of supported streams from 25 to 28. In each round, there are 20 random retrievals and eight prefetches. The reads from the disk cache to the host cache run in parallel with the $(p + v)$ disk accesses. In Round 1, the system supports the last five streams from an additional higher level cache buffer. When the requests for these five streams arrive—along with each stream’s first two blocks, which the “double-buffering” scheme needs—the multiple disk controller deposits the third block into the host’s cache. The server delays the data transmission for the five streams until all three blocks have been retrieved from the disk’s surface. Thus, higher level caches can issue disk retrieval requests for these five streams without experiencing any glitches. After Round 2 in Figure 2b, the host-level cache buffers required to implement this “triple buffering” are no longer needed and can be used by the new streams. Scheme S&P can be fully parameterized with v and p , always at the expense of disk cache memory size.

In this example, we assume that there is a simplified disk drive with a transfer rate of 10 Mbytes per second and a combined seek and rotational fixed overhead of 15 milliseconds for each request. We further assume that there is a constant display rate of 2 megabits per second and a round length of 1 second, which implies a block size of 250 Kbytes. Thus, at most, the disk head can read 25 blocks within a round, while the ratio of randomly retrieved blocks that can be “exchanged” (in terms of time) with prefetched blocks is $5/8$ (40 millisecond and 25 millisecond retrieval times, respectively).

GRADUAL PREFETCHING

The S&P scheme services streams without prefetching until the system reaches its maximum throughput. It then switches into the “prefetch” mode and, by exploiting the higher level caches, increases the maximum throughput without creating glitches. However, this scheme has a drawback: It requires three cache buffers for each stream, and thus each stream’s block will be skipped in

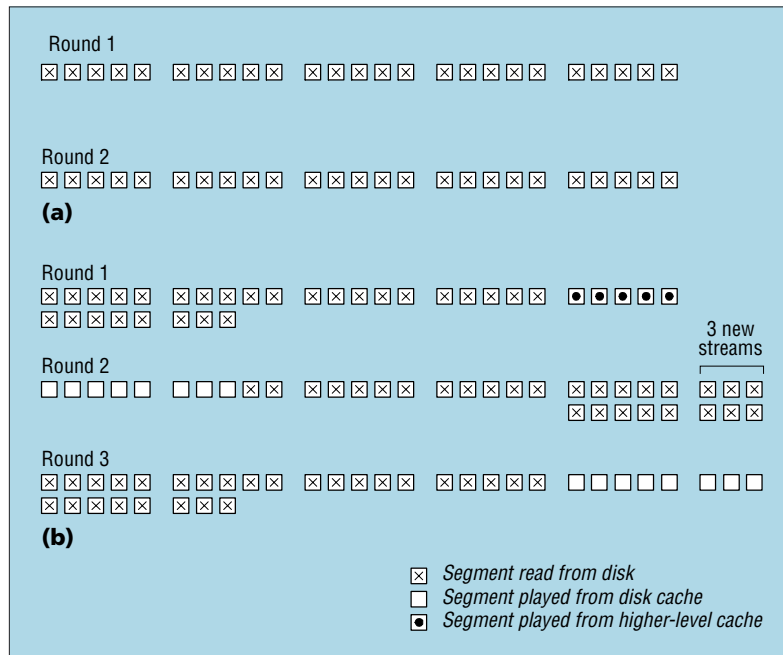


Figure 2. Scheme Sweep & Prefetch increases throughput by prefetching. Each box represents a particular video block. A box under another box represents adjacent blocks in the same stream. Two adjacent boxes in the same line represent blocks of two streams that the lower-level scheduler will schedule within a round in the same order as they appear (such as in SCAN order). In the top two rounds (a), $N = 25$, $v = 20$, and $p = 0$; in the bottom three rounds (b), $N = 28$, $v = 20$, and $p = 8$.

some round, creating extra start-up latency. We overcome this drawback using the Gradual Prefetching Scheme.

In gradual prefetching, we force the media server to function under the Scheme S&P all the time, regardless of the number of concurrent streams being served. At any given time, the disk heads therefore prefetch half of all supported streams. For every two newly admitted streams, one of them will have its next block prefetched during the first round. When the number of supported streams reaches the maximum, the Gradual Prefetching Scheme behaves like Scheme S&P. However, this time, half of the streams already have their next blocks in the disk cache; if they are skipped in the next round, no glitches will occur.

Gradual prefetching works with the maximum number of supported streams. Therefore, there is always enough time within a round to prefetch an additional

block for half of all new streams. Given this, no stream will experience extra start-up latency, and we no longer need triple buffering.

GROUPED PERIODIC MULTIROUND PREFETCHING

The Grouped Periodic Multiround Prefetching algorithm temporarily stores prefetched blocks in the host’s cache. Although disk caches and caches in other levels can store many of these blocks, we analyze the GPMP technique using only RAM so that we can directly compare our technique to other multimedia retrieval schemes with memory management that do not exploit prefetching.

Scheme GPMP introduces the concept of an *epoch*. The time interval of an epoch (or *virtual round*) is the total duration of a fixed number of actual rounds. GPMP’s macroscopic behavior is similar to other traditional grouped schemes^{3,5} if their

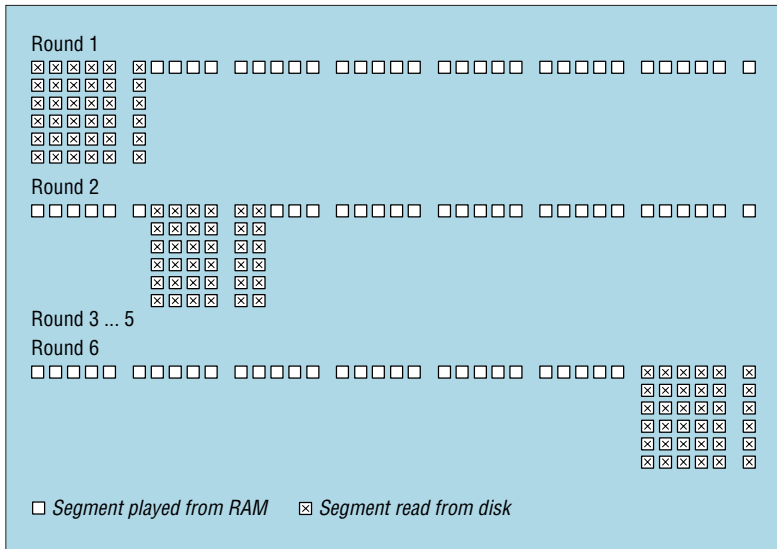


Figure 3. Grouped Periodic Multiround Prefetching temporarily stores prefetched blocks in the host’s cache so that they don’t have to be retrieved from the disk each round. Here, Round 1 corresponds to the server working at maximum throughput. $N = 36$, $v = 6$, $u = 5$, and $p = 30$.

Table 1. Simulated disk parameters.

DISK-DRIVE PARAMETER	VALUE
Capacity (Mbytes)	10,886
Heads/recording surfaces	18
Cylinders	6,720
Sectors per track	140 to 220
Sector size (bytes)	512
RPM	10,000
Seek function ¹ (msec)	$seek(d) = \begin{cases} 1.867 \cdot 10^{-3} + 1.315 \cdot 10^{-4} \sqrt{d}, & d < 1,344 \\ 3.8635 \cdot 10^{-3} + 2.1 \cdot 10^{-6} d, & d \geq 1,344 \end{cases}$
Average seek time (msec)	8
Average transfer rate (MBps)	14
System bus transfer rate (MBps)	40 (Ultra SCSI)

1. We can express the head-positioning time delays as a nonlinear function of d , where d is the number of consecutive cylinders that the head must traverse.

round duration is set equal to a GPMP epoch. However, GPMP’s reliance on prefetching results in a different functionality. At the system level, GPMP offers finer grained disk requests per stream, more flexible configuration, and higher service quality, because GPMP has lower start-up latencies.

During a GPMP round, the media server serves all streams—that is, the server delivers all blocks that sustain playback of the supported streams to the host application, which manages the data flow. The key idea is that these blocks don’t have to be retrieved from the disk each round; some of them are in the cache.

Thus, during a round, only a group containing a fraction (v) of the supported stream is randomly retrieved, while enough time remains in the round for u prefetches for each stream (where u represents the number of blocks prefetched within the same stream).

In the next round, the blocks that sustain the playback of the next group—which also contains v streams—are read from the disk surface along with the u prefetched blocks for each v stream (the streams within a round, as well as the different streams in subsequent rounds, are being served as before, in a SCAN manner). All streams will be served during

each round because the $N - v$ blocks, which were not retrieved from the disk during the current round, are in the cache, having been prefetched in previous rounds (N being the total number of streams supported). The u prefetched blocks of a stream read during a given round sustains its playback from the cache for the next u rounds. That is, after u rounds, the same streams read from disk during a given round have to be read from disk again.

The epoch is $k = u + 1$ rounds. The total number of streams supported under GPMP is $N = v(u + 1)$. Figure 3 demonstrates GPMP using the same framework as the previous figures. The maximum number of streams supported at $u = 5$ is 36 ($v = 6$). Gradual prefetching strategies ensure that during Round 1 in Figure 3, the 30 streams not served from disk have their blocks already stored in the cache as a result of prefetching during the previous rounds. More specifically, because we assume that Round 1 corresponds to the server working at maximum throughput, each of the second group of six streams has one remaining block in the cache, each of the third group of six streams has two blocks in the cache, and so on. “Played” blocks are immediately discarded.

Performance evaluation

We analytically evaluated the performance of our algorithms. For space reasons, we present here our results from a simulation, which closely matches the results from our formal analysis. We simulated a continuous-media server that includes the retrieval of continuous media streams from secondary storage into RAM through a system bus, and a pregenerated workload equivalent to that of a typical server use. To realize our simulation, we used the DiskSim Simulation Environment.⁹ However, DiskSim does not address some onboard cache management issues that newer disk drives handle and that are essential for implementing our algorithms. These issues are the ability to enable and disable requested-data caching, so that the cache can be replaced in various ways; the abil-

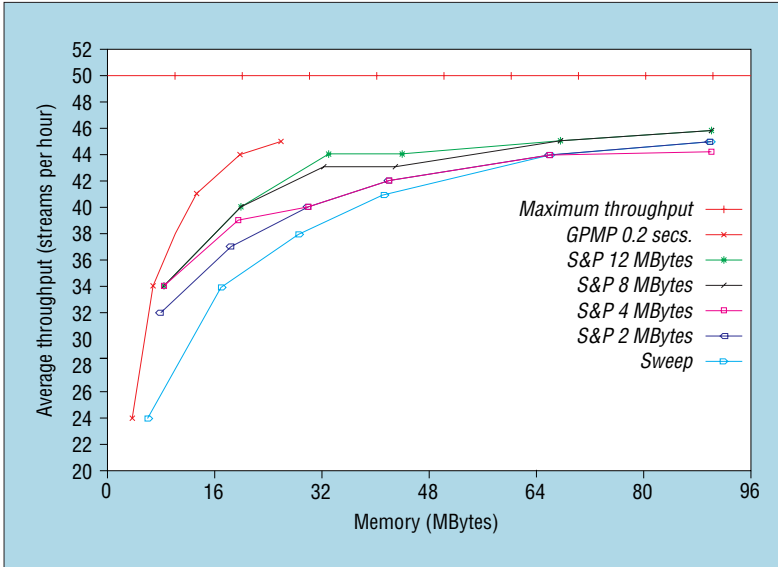


Figure 4. Throughput performance versus memory requirements for prefetching with Sweep & Prefetch algorithms and Sweep.

ity to switch prefetching off and on at various depths; and the ability to change the disk cache segmentation. We added these capabilities by slightly modifying DiskSim’s source code.

THE EXPERIMENT

As our simulation example, we use a video library that holds videos that are 90 to 120 minutes long. We order the videos according to popularity, and they follow a Zipfian distribution. Our trace generator considers newly arrived videos with a Poisson distribution, and we assume disk partitioning: large portions of the video are placed onto disk successively. We implemented the S&P family of algorithms in DiskSim using the trace generator to pass hints and evaluate our algorithms and Scheme Sweep.

Because the DiskSim software release had outdated disk models, we assumed another disk model (see Table 1). We set the seek-function values and those of various delays and overheads to equal those on currently available disk drives.

RESULTS

We tested the Sweep and S&P schemes for their maximum achievable throughput without glitches for disk cache sizes of 2, 4, 8, and 12 Mbytes and for rounds of 0.5, 1, 1.5, 2, 3, and 4 seconds. We configured all schemes to use the same amount of main memory for each round length. The extra gains in

throughput for the S&P algorithms came from exploiting the onboard cache, which traditional techniques such as Sweep cannot use. When we compare our prefetching strategies to Sweep, we see 20 to 70% improvements in throughput for round lengths of 0.5 to 1.5 seconds and disk cache sizes of 2 to

12 Mbytes. For longer rounds, all S&P configurations and Sweep converge toward an upper bound. However, as Figure 4 shows, longer round lengths have little to no practical use, as they explode memory requirements and create undesirable start-up latency, especially for short media clips.

As Figure 4 shows, GPMP outperforms the traditional Sweep algorithm, achieving higher throughput using the same memory, but shows poor start-up latency. Given the round length, we can adjust GPMP’s parameters v and u , trading off increased throughput for low start-up latency. Also if we set $u = 0$ and choose an appropriate value for v , GPMP’s performance equals Sweep’s. In either case, the maximum throughput is $[(u + 1) \cdot v]$ or simply $k \cdot v$. As intuition suggests, higher k values increase queuing delays and hence create higher start-up latencies for incoming requests.

Figure 5 shows the performance of a single disk media server with round

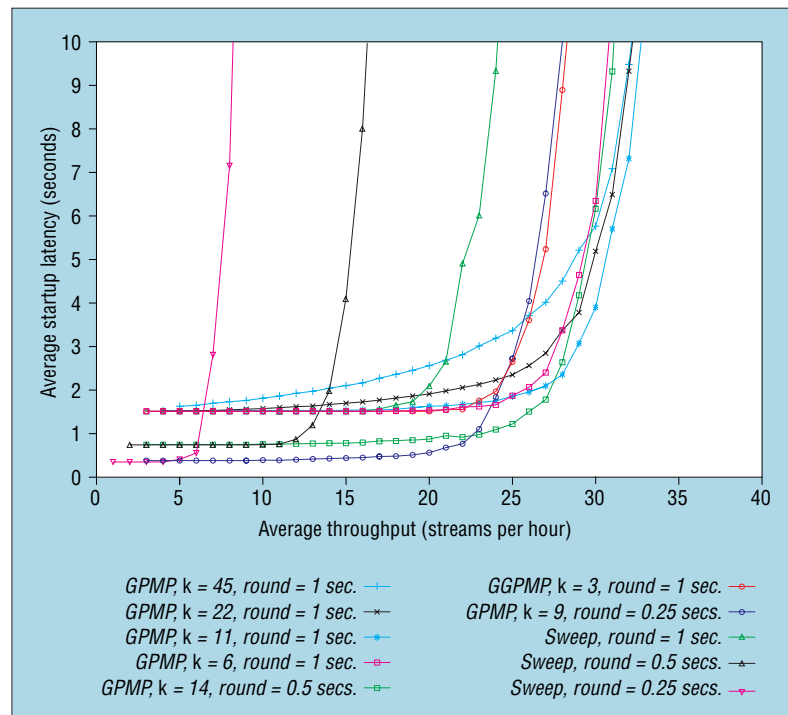


Figure 5. Throughput performance versus start-up latency for the Grouped Periodic Multiround Prefetching and Sweep algorithms with different k values and round durations.

durations from 0.25 to 1 seconds. For low request-arrival rates, lower values of k induce lower start-up latencies. As the request-arrival rate increases, queuing delays occur due to the server's high workload, and GPMP configurations for higher throughput become more beneficial for start-up latency.

Certain configurations with different round lengths can yield the same throughput, as long as parameter k is tuned appropriately (for example, when round length is 0.25, k is 19; when round length is 1, k is 6). In this case, we prefer the configuration with the lower round length duration, because it has lower start-up delays.

OUR PREFETCHING strategies introduce significantly higher maximum throughput for disks compared to the traditional Sweep strategy for retrieving continuous media blocks—as high as 60 to 70% more for a 500 millisecond round. We also found that Sweep could not exploit the large, on-board buffer we used in our simulation.

To our knowledge, disk manufacturers have yet to publish any prefetching techniques designed for continuous media retrievals that also account for the concept of a round. Furthermore, interfering in the prefetching strategy led to the expected benefits: requests found in cache (hits) can be transferred by I/O controllers to the host in parallel to other, disk surface-to-buffer transfers.

The current technology trends suggest that our techniques will show even better results for future disk products, because transfer rates will continue to improve (at a much faster pace than seek delays), and more powerful controllers operating on even bigger embedded caches are certain to follow. //

ACKNOWLEDGMENTS

The Exapsis project, which is funded by the General Secretary of Research and Technology, Greece, supported our research.

References

1. D.J. Gemmel et al., "Multimedia Storage Servers: A Tutorial," *Computer*, Vol. 28, No. 5, May 1995, pp. 40–49.
2. E. Riedel, G. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia Applications," *Proc. Int'l Conf. Very Large Databases (VLDB)*, Morgan Kaufmann, San Francisco, 1998.
3. E. Chang and H. Garcia-Molina, *Effective Memory Use in a Media Server*, (extended version), Stanford Tech. Report SIDL-WP-1996-0050; www.diglib.stanford.edu/cgi-bin/get/SIDL-WP-1996-0050 (current July 2000).
4. B. Ozden, R. Rastogi, and A. Silberschatz, "Disk Striping in Video Server Environments," *Proc. Int'l Conf. Multimedia Computing and Systems (ICMCS)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996.
5. P.S. Yu, M.S. Chen, and D.D. Kandlur, "Grouped Sweeping Scheduling for DASD-Based Multimedia Storage Management," *ACM Multimedia Systems*, Vol. 1, No. 2, 1993, pp. 99–109.
6. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46–61.
7. P.J. Denning, "Effects of Scheduling on File Memory Operations," *Proc. American Fed. Information Processing Societies Conf. (AFIPS)*, Thomson Book Company, Washington, D.C., 1967, pp. 9–21.
8. Y. Robogianakis et al., "Disk Scheduling for Mixed-Media Workloads in Multimedia Servers," *Proc. Sixth ACM Int'l Multimedia Conf.*, ACM Press, New York, 1998.
9. G.R. Ganger, B.L. Worthington, and Y. N. Patt, *The DiskSim Simulation Environment Version 1.0 Reference Manual*, CSE-TR-358-98, Univ. of Michigan, 1998; www.eecs.umich.edu/techreports/cse/1998/CSE-TR-358-98.pdf (current July 2000).

Stavros Harizopoulos is a PhD student in computer science at Carnegie Mellon University and a member of the Parallel Data Laboratory. He studied electronics and computer engineering at the Technical University of Crete, Greece. His research interests are in high-performance storage systems, with an emphasis on streaming media applications. He is a Lilian Voudouri Foundation fellowship recipient. This work is based upon his undergraduate diploma thesis, under the supervision of Peter Triantafillou. Contact him at the Computer Science Dept., Carnegie Mellon Univ., Pittsburgh, PA 15213-3891; stavros@cs.cmu.edu.

Costas Harizakis is a graduate student in electronics and computer engineering at the Technical University of Crete. He received his diploma in electronics and computer engineering from the Technical University of Crete. His research interests are in high-performance intelligent storage systems and distributed proxy caches supporting multimedia applications. Contact him at the Dept. of Computer Engineering, Technical Univ. of Crete, Chania, Greece; harizak@speech.gr.

Peter Triantafillou directs the Software Systems Engineering and Network Applications Laboratory at the Department of Electronic and Computer Engineering at the Technical University of Crete, Greece. He has a PhD from the Department of Computer Science at the University of Waterloo. His research efforts are in high-performance multimedia computing and networking. He has published extensively in these and other areas and has served on the DEXA '2000, EDBT '2000, ACM MobiDE '99, ACM SIGMOD '99, FODO '98, RIDE '98, and EDBT '98 conference program committees and as a reviewer for many international journals. Contact him at the Dept. of Computer Engineering, Technical Univ. of Crete, Chania, Greece; peter@ced.tuc.gr.